ANGLIA RUSKIN UNIVERSITY

FACULTY OF SCIENCE & TECHNOLOGY

VWADM: AN ARCHITECTURE-INSPIRED METHOD FOR DESIGNING VIRTUAL WORLD APPLICATIONS

WILLIAM AUGUSTUS SAWYERR, JNR.

A thesis in partial fulfilment of the
requirements of Anglia Ruskin University
for the degree of Doctor of Philosophy

Submitted: July 2018

For M. Lois LaPlante

Acknowledgements

Academic acknowledgements. I would like to express my profound gratitude to my supervisors Dr. Cristina Luca, Dr. George Wilson, Dr. Mike Hobbs and Dr. Silvia Cirstea (Adviser) for all their help with the research and with this thesis.

I am also very grateful to Prof. Marcian Cirstea for all his help and advice over the years.

Finally, I would like to thank Prof. Sergiu Dascalu and Dr. Shabnam Sadeghi-Esfahlani for agreeing to examine the thesis and their valuable feedback with regards to improving it.

Technical acknowledgements. Some of the inspiration for the research that this thesis is based on comes from my experiences in the worlds of several virtual reality (VR) platforms such as Active Worlds, Second Life, Open Simulator and Sansar. These platforms also provided me with the space and tools for conducting my experiments, including the modelling of VW applications and coding dynamic and interactive behaviours using Java, C, the Linden Scripting Language (LSL) and C#. In particular, Active Worlds provided me with the testing ground for GDAs in conjunction with Jess.

Many thanks to Prof. Ning Gu (UniSA), Dr. Gregory Smith (CSIRO) and Prof. Mary Lou Maher (UNC Charlotte) for their assistance with the implementation of the GDG framework and GDA model.

ANGLIA RUSKIN UNIVERSITY

ABSTRACT

FACULTY OF SCIENCE & TECHNOLOGY

DOCTOR OF PHILOSOPHY

VWADM: AN ARCHITECTURE-INSPIRED METHOD FOR DESIGNING VIRTUAL WORLD APPLICATIONS

WILLIAM AUGUSTUS SAWYERR, JNR.

July 2018

Virtual World (VW) applications are digital 3D-simualtion environments used for military, education, sport and healthcare purposes. In this work, an architecture-inspired method called the Virtual World Applications Design Method (VWADM) is presented for designing such applications. The VWADM combines a generative design grammar (GDG) framework and a generative design agent (GDA) model to form the template of a mechanism that can be used for creating place models of VW applications during the design process. The GDG framework is used to develop GDGs for capturing spatial data that pertains to VW applications, transforming and storing such data and subsequently using it to create place-oriented conceptual models of these applications. In complement, the GDA model is used to develop GDAs for automatically generating physical models of VW applications, whose specifications are derived from the GDGs.

The development of the VWADM is grounded in design science research (DSR), which is a research paradigm that provides a set of synthetic and analytical techniques and perspectives for understanding, conducting, evaluating and reporting creative work. DSR involves answering questions that are relevant to human problems through the design of novel and innovative artefacts.

The thesis focuses on describing the design of the VWADM, demonstrating its functionality and evaluating its fitness and utility for designing VW applications. Details are presented of software that was developed to establish proof of concept for the VWADM. Through the development of the VWADM, the research establishes (1) a set of modelling concepts that use grammar and rules as the basis for capturing the semantics of the conceptual models of VW applications, (2) a set of visual notations that include basic shapes and symbols, which can be used to present the conceptual models of VW applications to stakeholders for them to manually review, (3) an intelligent component that is used for automatically generating physical models of VW applications and (4) a stepwise process, which involves layout design, object design, navigation design and interaction design and acts as a guide for the process of creating the conceptual and physical models of VW applications.

Feedback from developers suggest that the VWADM can provide a significant contribution to automating the process of VW design specifically from the point of view of place-oriented design. Some ideas for future work are presented.

Keywords: architecture-inspired design methods; place-oriented design; vw applications

Table of Contents

List of Figures

List of Tables

List of Publications

This thesis contains ideas and work that have been published in the following:

Sawyerr, W., 2016. An Approach for Designing Applications in 3D Virtual Worlds. In: ACM (Association for Computing Machinery), 8th International Workshop on Massively Multiuser Virtual Environments. Klagenfurt, Austria, 10-13 May 2016. New York: ACM.

Sawyerr, W., 2016. An Application of the Design Science Research Theoretical Framework and Methodology in the Construction of an Approach for Designing Applications in 3D Virtual Worlds. Journal of Advances in Information Technology, 7(4), pp.259-263.

Sawyerr, W. and Hobbs, M., 2014. Designing for Usability in 3D Virtual Environments. In: IFIP (International Federation for Information Processing), 2nd International Workshop on Usability and Accessibility Focused Requirements Engineering. Karlskrona, Sweden, 25 August 2014. Piscataway: IEEE.

Sawyerr, W., Brown, E. and Hobbs, M., 2013. Using a Hybrid Method to Evaluate the Usability of a 3D Virtual World User Interface. International Journal of Information Technology & Computer Science, 8(2), pp.66-74.

Holley, D., Howlett, P., Hobbs, M. and Sawyerr, W., 2013. 'The Chaotic Science Lab': Supporting Trainee Science Teachers - A Cross Departmental Project. Networks, 16, pp.51-58.

Sawyerr, W.A. and Pinkwart, N., 2011. Extending the Private and Domestic Spaces of the Elderly. In: CSCW (Computer Supported Cooperative Work), Workshop on Socialising Technology Among Seniors in Asia. Hangzhou, China, 19-23 March 2011. New York: ACM.

Sawyerr, W.A. and Pinkwart, N., 2011. Designing a 3D Virtual World for Providing Social Support for the Elderly. In: ECSCW (European Conference on Computer Supported Cooperative Work), Workshop on Fostering Social Interactions in the Ageing Society. Aarhus, Denmark, 24-28 September 2011.

Copyright

1. Introduction

This thesis is about designing virtual world (VW) applications. Section 1.1 discusses the research problem and motivation for the work. Section 1.2 provides definitions of some of the concepts and terms that are used in this thesis in relation to virtual reality (VR), as well as a brief definition of the scope of the thesis. The intention is to form a common understanding of the meanings of these concepts and terms as they are used in the context of this thesis and to delimit the area of the VR spectrum that concerns it. Section 1.3 discusses the aims and objectives of the research. Section 1.4 provides a statement of the research hypothesis. Section 1.5 highlights the contributions to knowledge. Section 1.6 provides a summary of the approach that was used to conduct the research. Finally, Section 1.7 provides a brief description of the organisation of the rest of the thesis.

1.1. Statement of the Research Problem and Motivation

The rate at which VR is being commoditised these days far exceeds the corresponding rate at which methods are being developed in software engineering (SE) for designing the virtual experiences that are promised by this technology (Johnston, et al., 2017; Ullrich & Kunkler, 2018). So far, developers have relied on the use of existing methods in SE such as the structured (Constantine & Yourdon, 1979; Jacobson, et al., 1993), behavioural (DeMarco, 1978; Ward & Mellor, 1985), formal (Hall, 1996; Wordsworth, 1996), interactional (Moggridge, 2007; Sharp, et al., 2007) and iterative (Boehm, 1988; Nielsen, 1992) methods for creating models of VW applications during the design process (Hong, 2015; Dawson, et al., 2017; Sutcliffe, et al., 2018). These methods are based on certain viewpoints that determine the types of such models. For example, structured methods are based on a

structure-oriented viewpoint that is used to create object models of VW applications[1]. Methods in SE are expected to represent the different viewpoints that stakeholders have for describing VW applications (Sommerville, et al., 1998; Rao & Prasad, 2012). However, user activity in VWs establishes a place-oriented viewpoint (Witmer, et al., 1996; Churchill, et al., 2001), which puts special requirements on them in terms of case-specific knowledge such as patterns, conventions and domain experience. Therefore, SE requires methods that support this viewpoint and can be used by developers for modelling VW applications during design.

In order to address the issue, this thesis proposes a new method called the virtual world applications design method (VWADM), which was developed for designing VW applications (Sawyerr, 2016). The VWADM is based on a place-oriented viewpoint and combines a generative design grammar (GDG) framework and a generative design agent (GDA) model to form the template of a mechanism that developers can use to create place models of VW applications during the design process. The GDG framework is used to develop GDGs for capturing spatial data that pertain to VW applications, transforming and storing such data and subsequently using it to generate conceptual models of these applications. In complement, the GDA model is used to develop GDAs for automatically generating physical models of VW applications, whose specifications are derived from the GDGs.

The research that this thesis is based on is motivated by place design in architecture. Place design is the process of arranging objects and spaces to create an environment that supports certain activities (Tepavcevic, 2017). Fig. 1.1 shows an example of the arrangement of some objects and space to create a place for buying and eating food (and socialising while doing so).

---

[1] Structured methods were invented in the 1970s to support functional design (Finkelstein, et al., 1992). Therefore, it is also possible for them to be based on a function-oriented viewpoint, which is used to create function models of VW applications such as data flow models.

Figure 1.1. The arrangement of some objects in a space to create a restaurant (Street Kitchen, 2017)

Place design involves the borrowing of function, form and conception from precedents, symbols, metaphors and analogies (Hattenhauer, 1984). This approach is referred to as designing from patterns and based on the assumption that a combination that proved to be operational in earlier circumstances will, with proper adjustments made to fit a new context, continue to be so for the same circumstances.

According to (Kalay & Marx, 2003), if no appropriate precedent can be found, symbols, metaphors and analogies may be used to engender some inherent quality that is found in the original pattern[2]. An example of this approach is Santiago Calatrava's L'Hemisfèric auditorium in the City of Arts and Sciences complex in Valencia (see Fig. 1.2).

[2] (Kalay & Marx, 2005) further argue that in case an appropriate precedent cannot be found, new forms could be developed, which may acquire their own status as precedents.

Figure 1.2. The L'Hemisfèric auditorium in the City of Arts and Sciences complex in Valencia (Tur, 2005)

Place design in architecture can be considered a visual art that is similar to painting and sculpture (Tillich, 1987). As such, it requires the manipulation of certain art elements to create a unified whole. Place design requires architects to prepare sketches (see Fig. 1.3) and then draw and refine the form (of the areas of function) of a new place.

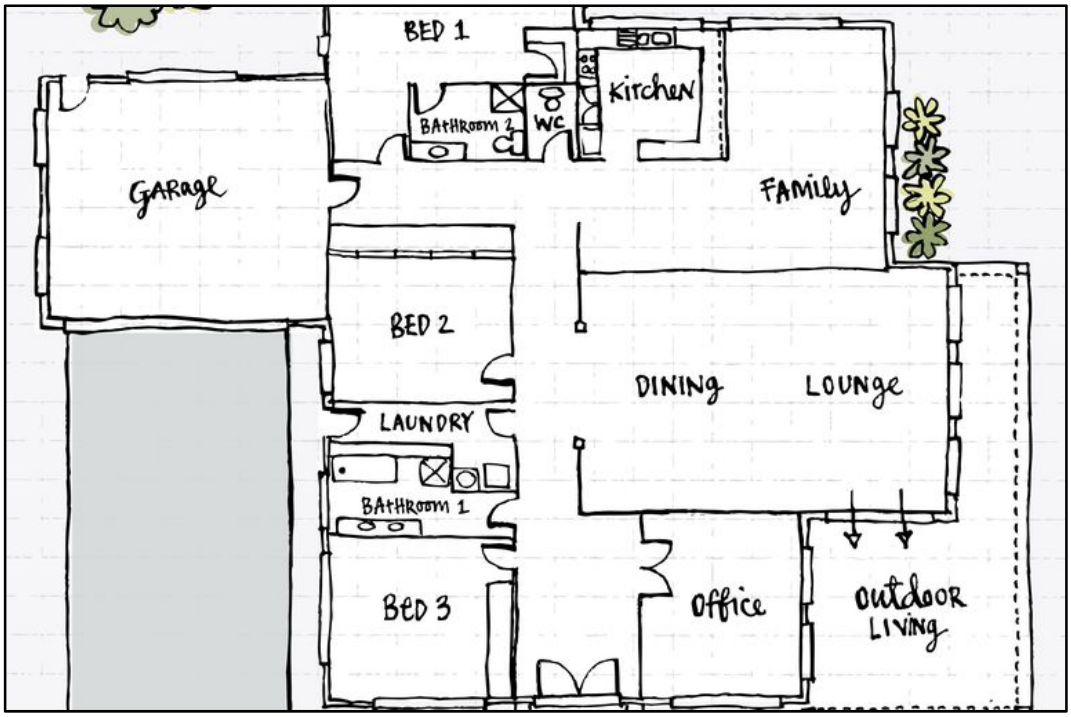


Figure 1.3. The sketch of a new place during design (Craven, 2017)

The sketch of a place is comprised of sensory elements such as lines, shapes, colour and texture. These elements are combined during place design to make formal compositions that create a certain style (i.e., pattern, rhythm, symmetry, balance, contrast, proportion, theme, and unity)[3]. Fig. 1.4 shows an example of the sketch of a place with reference to its sensory elements.

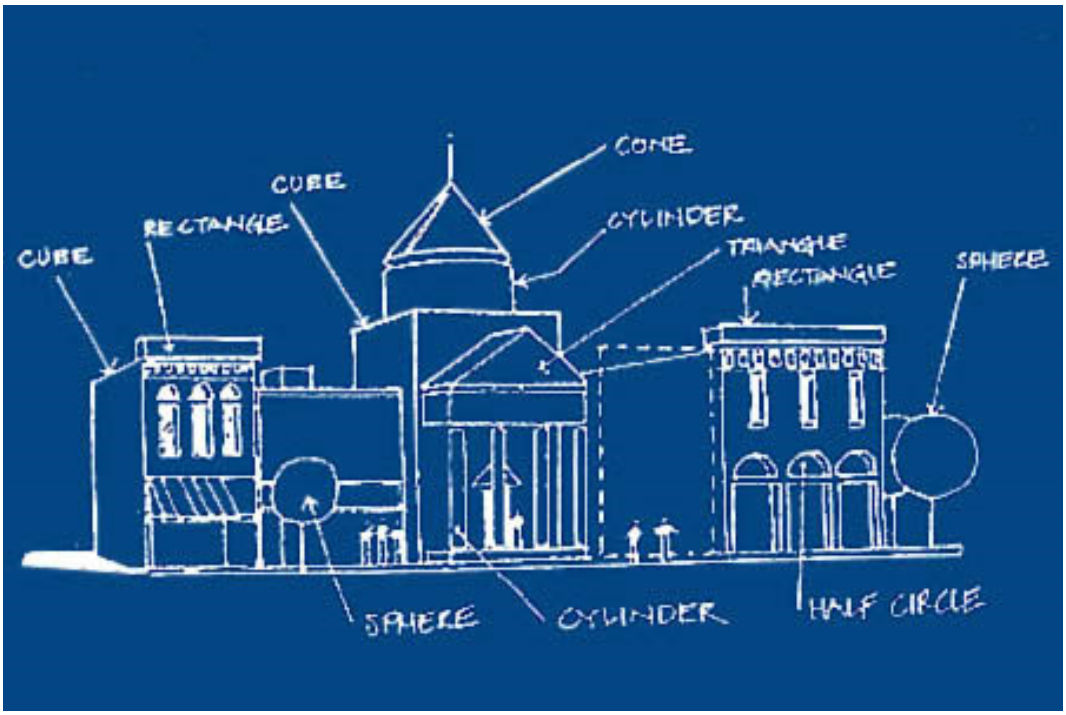


Figure 1.4. The sketch of place and some of its sensory elements (The Center for the Study of Art and Architecture, 2002)

The sensory elements of a place are not restricted to sketches only. Instead, they are assimilated in the design and manifest in the actual construction of the place. Fig. 1.5 shows the example of a place that exhibits sensory information in the form of colours, lines, triangles, circles and rectangles.

[3] See (Stebbing, 2004) and (Barrett, 2011) for a further discussion on visual composition in architecture.

Figure 1.5. A place that exhibits sensory information in the form of colours, lines, triangles, circles and rectangles (Craven, 2018)

1.2. Definitions and Scope of the Research

VR is a collection of technologies such as images, sound, sensors, trackers, central processing units (CPUs) and graphic processing units (GPUs) and techniques such as computer-generated imagery (CGI), which are used to replicate a real environment or create an imaginary one (Isdale, 1993; Vince, 1998). The replicated or created environment is usually referred to as a simulator or a virtual environment (VE). A VE is an artificial container that could take on many forms from as low-level as a hypervisor in which a virtual machine (VM) runs (Graziano, 2011) to mid-level as fixtures for overlaying augmented sensory information on a workspace (Rosenberg, 1992) and high-level as an environment that is modelled after our universe (Sylvain, et al., 2010; Hyndman, 2011; Sherstyuk & Treskunov, 2014). Consequently, a VE may be comprised of space, time, energy and physical laws. In addition, similarly to our universe, a VE may include inhabitable worlds, which inherit global properties from the VE (e.g. stars, wind and gravity) and have their own specialised properties (e.g., day-night cycles, weather patterns, people, places, flying and teleportation). These inhabitable worlds are usually referred to as virtual worlds (VWs). A

VW is an environment that is a unit of organisation in a VE (Gertrudix & Gertrudix, 2012; Tirpak & Ramic, 2014). It is the space in which people experience VR either directly (i.e., by self) or by proxy (i.e., using avatars). When people create places in VWs to support their various in-world activities, those places are referred to as VW applications (Mallempati, et al., 2010). By token of the different levels of sophistication of the VEs in which they run (or otherwise by deliberate design choice), VWs may be text-based, two-dimensional (2D), Two-and-a-half dimensional (2.5D), three-dimensional (3D) or experimental n-dimensional (nD) environments and take on different forms such as console-based applications, fantasy worlds or real earth-like worlds.

VWs are rendered on to visual display units (VDUs) such as mobile phone or tablet screens, laptop or desktop computer monitors, projection screens, head-up displays (HUDs) and helmet or head-mounted displays (HMDs) so that people can be able to access them. VDUs are usually integrated with other components such as sensors, trackers, haptic devices and control or management software to form a VR system. A VR system is the amalgamation of various input-processing-output (IPO) components to form a compact device (or platform) that enables people to consume VR (Burdea, et al., 1996). VR systems may exist in a standalone configuration such as mobile or wearable VR systems (Henrysson, et al., 2005; Barfield, 2016), desktop VR systems (Tait, 1992) and cave automatic virtual environment (CAVE) VR systems (Cruz-Neira, et al., 1993). Otherwise, they may be used as an enabling technology for a diverse range of other systems such as cue weapons systems (Haywood, 1995), situation awareness systems (Foyle, et al., 1996), medical scanners (Deshmukh, et al., 2015), robots (Guerin & Hager, 2016) and the web (Vukicevic, et al., 2016), essentially forming a system of systems. Fig. 1.6 shows the architecture of a generic VR system, which includes a VE and some VWs.

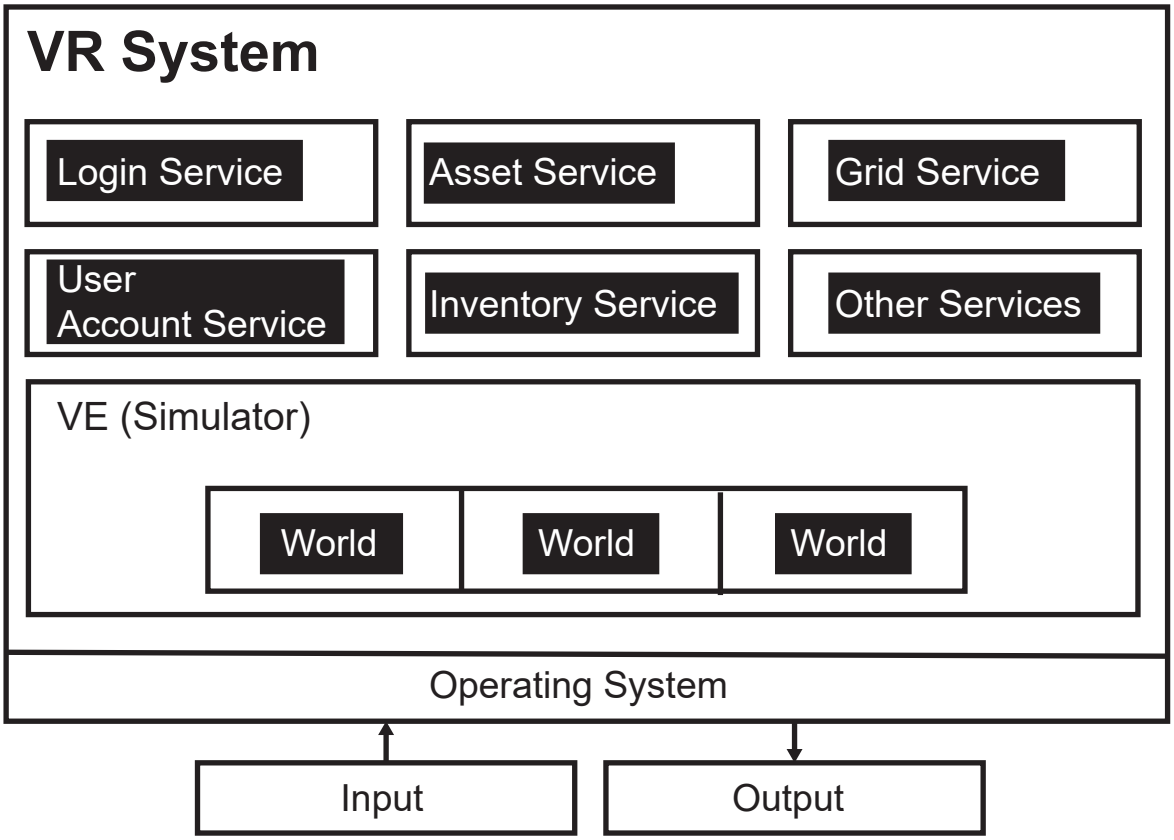


Figure 1.6. A generic VR system (OpenSimulator, 2017)

VR systems can be classified in a number of ways. For example, they can be classified according to their platform (i.e., mobile or wearable, desktop-based or CAVE). They can also be classified according to where they fall on the reality-virtuality (RV) continuum (see: Fig. 1.7). The RV continuum is a classic scale that is used to help conceptualise all possible compositions of VR systems (Milgram, et al., 1994). It is organised as follows: the extreme left side of the scale represents reality and the extreme right side of the scale represents VR. All VR systems that fall in-between both extremes are referred to as mixed-reality (MR) systems, which are systems that merge reality and VR (Milgram & Kishino, 1994). Examples of MR systems are augmented reality (AR) systems and augmented virtuality (AV) systems. The RV continuum takes into account the extent to which the various VR systems replicate the real environment (i.e., the real world). As such, the extreme left side of the scale represents the real world and the extreme right side of the scale represents a VE. All VR systems that fall in-between both extremes are capable of generating an environment that has aspects of both the real world and a VE (Zhu, et al., 2016). A final overarching point about VR systems is that they can be classified as immersive or non-immersive systems.

An immersive VR system is one that enables a person to feel as though they are physically present in a VW, usually through the use of stereoscopic VDUs such as HMDs (Van Dam, et al., 2000). On the contrary, a non-immersive VR system puts less emphasis on immersion and instead enables people to interact with its VWs by proxy (Obeysekare, et al., 1996), usually through the use of monoscopic VDUs such as mobile phone or tablet screens.

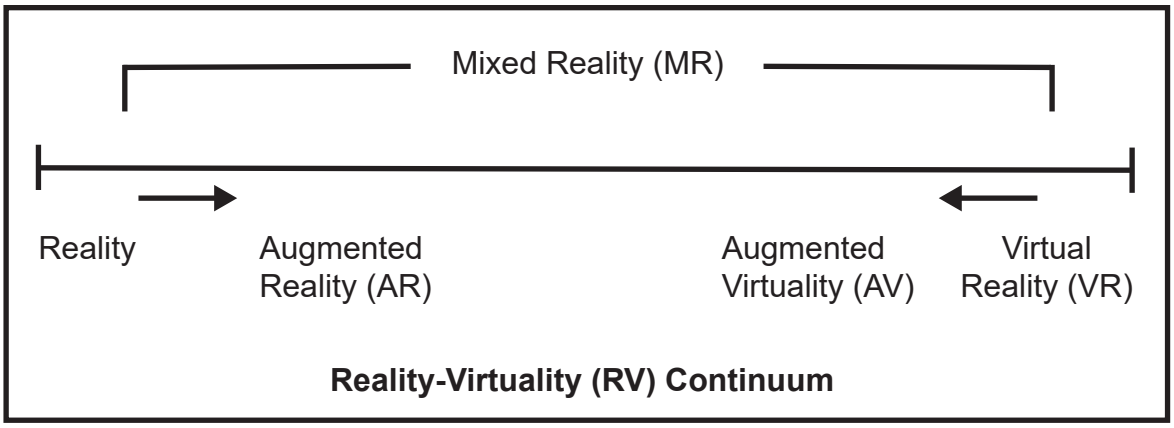


Figure 1.7. RV continuum (Milgram, et al., 1994)

The scope of this thesis is limited to the case on the extreme right side of the RV continuum (i.e., VR). Specifically, the research that this thesis is based on deals with non-immersive 3D desktop-based VR systems, which provide an inexpensive and easily accessible platform for exploring the design of virtual experiences. Server simulation software for desktop-based VR systems is mostly free and runs easily on personal computers (PCs) without requiring any special hardware besides a modest amount of memory. Similarly, the client software can also run easily on PCs without the need for any special hardware besides a medium range GPU. In addition, the client software is portable across multiple low-cost digital devices such as mobile phones and tablets, enabling VR that is produced primarily for PCs to be accessed using other comparably cheaper and ubiquitous devices.

1.3. Aims and Objectives

The aim of the research was to develop an architecture-inspired method for designing VW applications, in which visuals will play an important role. The method is intended for use by developers to enable them to create place models of VW applications during the design process. As such, the main objectives of the research were as follows:

- To develop a method that is comprised of a set of tools that can be used for creating place-oriented conceptual models of VW applications.
- To develop a method that is comprised of a set of visual notations that can be used for representing the spatial elements that pertain to VW applications.
- To develop a method that is comprised of an intelligent component, which can be used for creating physical models of VW applications that are derived from the conceptual models of these applications.

1.4. Research Hypothesis

The aims and objectives of the research are based on the hypothesis that a method can be developed, which has the functionality for creating place-oriented conceptual and physical models of VW applications and be of use to developers for designing these types of applications (H1). The general hypothesis is usually difficult to evaluate as a whole (Offermann, et al., 2009). Therefore, it was refined into the following two sub-hypotheses:

- a method can be developed, which has the functionality for creating place-oriented conceptual and physical models of VW applications (H1a).
- a method can be developed, which is useful to developers for creating place-oriented conceptual and physical models of VW applications during design (H1b).

1.5. Contributions

The research develops the VWADM using techniques from architecture and applies it in SE as a novel solution to the problem of designing VW applications. In particular, the VWADM provides support for the place-oriented viewpoint and can be used for creating place models of VW applications during design. As such, the contributions of the research are highlighted in terms of the following aspects:

- The establishment of a set of modelling concepts that use grammar and rules as the basis for capturing the semantics of the conceptual models of VW applications.

- The establishment of a set of visual notations, which include basic shapes and symbols that can be used to present the conceptual models of VW applications to stakeholders for them to manually review.
- The establishment of an intelligent component that is used for automatically generating physical models of VW applications.
- The establishment of a stepwise process, which involves layout design, object design, navigation design and interaction design and acts as a guide for the process of creating the conceptual and physical models of VW applications.

1.6. Research Approach

The approach that was used for conducting the research involves the following three key phases:

Problem identification: literature reviews were conducted to gain an understanding of the state of the art in designing VW applications. In addition, information was collected in-world and from online forums dedicated to desktop VR systems about users and developers and some of the problems that affect their use of these platforms.

Solution design: theory and design and development knowledge related to linguistic design in VWs, worldviews for designing in VWs and agent-based design in VWs was used to determine the composition and functionality of the VWADM.

Evaluation: a user study (based on a survey of developers) was conducted to evaluate the features and capabilities of the VWADM with respect to their significance for practical application in designing VW applications.

Together, the above three phases (i.e., problem identification, solution design and evaluation) contribute to a summary of the results of the research, which is usually in the

form of publications such as this thesis itself. Fig. 1.8 shows an overview of the research approach.

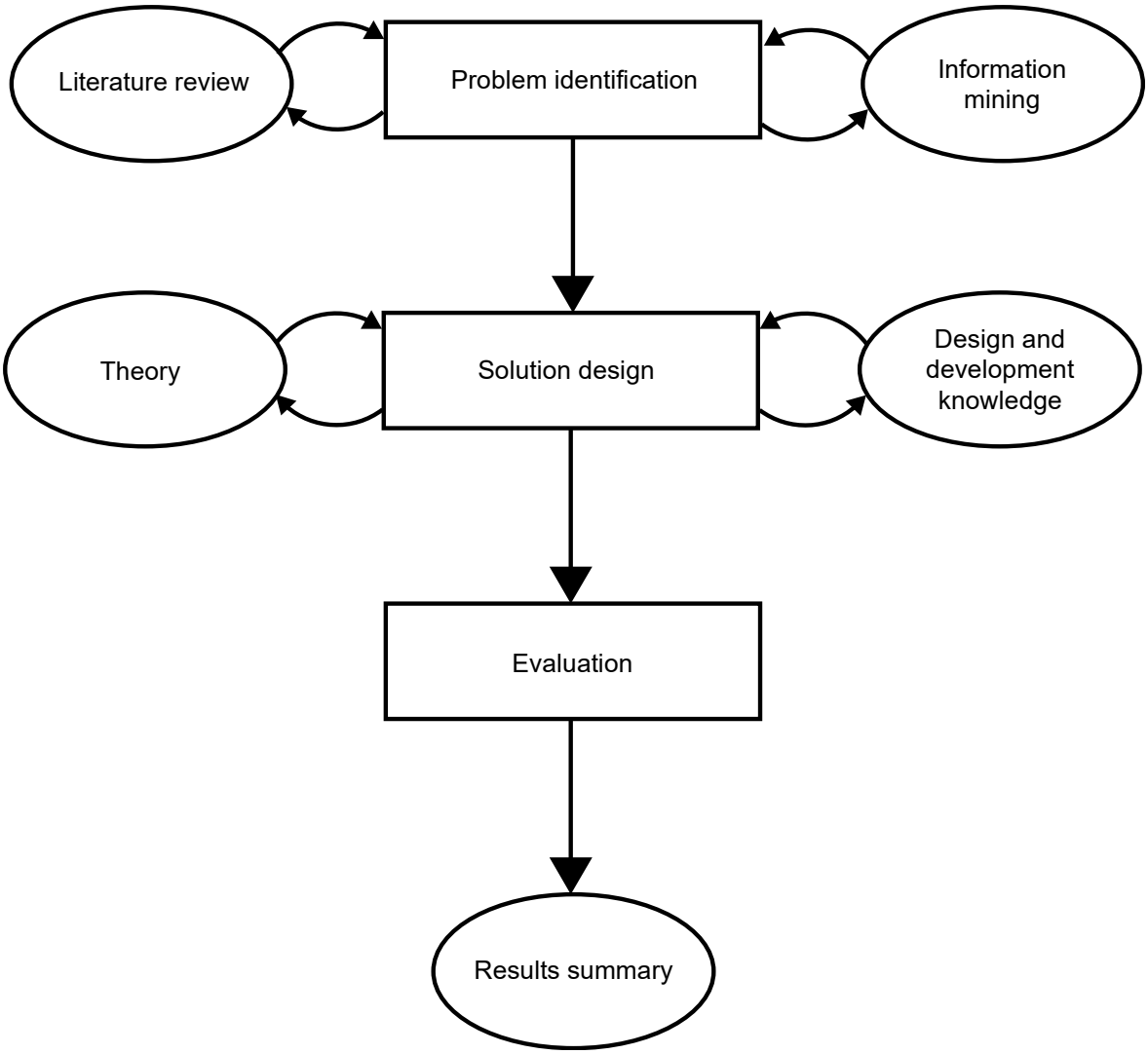


Figure 1.8. An overview of the research approach

The rectangles in the diagram above represent the key phases of the research approach. The ovals represent sub-processes of the research approach, which are also knowledge inputs and outputs for the key phases.

1.7. Organisation of the Thesis

The remainder of the thesis is organised as follows:

1.7.1. Literature Review

Chapter 2 presents the literature review, which discusses some of the immediate areas that influenced the development of the research in terms of the state of the art in designing VW applications. Using an SE lens, Section 2.1 provides a definition of design. Section 2.2 discusses a few of the reasons design is important, especially in the context of developing VW applications. Next, Section 2.3 reviews some examples of desktop-based VR systems, which provide platforms that accommodate and facilitate designing VW applications. Section 2.4 reviews some of the existing methods in SE that are currently used for designing VW applications. Section 2.5 briefly discusses the architectural concept of placeness. The chapter finishes in Section 2.6 with a brief examination of place-oriented modelling techniques in architecture.

1.7.2. Methodology

Chapter 3 discusses the design science research (DSR) paradigm, which was used for conducting the research. Firstly, Section 3.1 provides an overview of DSR. Next, Section 3.2 discusses the rationale for using it. This is then followed by Section 3.3, which provides an overview of the core principles of DSR. Section 3.4 discusses the DSR process model that was used for conducting the research and presenting it in this thesis. The chapter finishes in Section 3.5 with the review of a case study that was used as a pilot study for the research. The purpose of this review is to give an example of the search process that was used to arrive at the VWADM.

1.7.3. Design and Development of the VWADM

Chapter 4 discusses the design and development of the VWADM using theory and design and development knowledge related to linguistic design in VWs, worldviews for designing in VWs and agent-based design in VWs. Section 4.1 discusses linguistic design in VWs, Section 4.2 discusses views for designing in VWs and the chapter finishes in Section 4.3, which discusses agent-based design in VWs.

1.7.4. VWADM: A New Method for Designing VW Applications

Chapter 5 provides a detailed description of the VWADM, including its two components: the GDG framework and GDA model. Section 5.1 provides an overview of the composition of the VWADM. Next, Section 5.2 provides details of the composition of the GDG framework. This is followed by a discussion of the details of the composition of the GDA model in Section 5.3, which is also where the chapter finishes.

1.7.5. Implementation of the VWADM

Chapter 6 provides an overview of software called the Active Worlds (AW) agent package, which was developed to serve as a proof of concept for the VWADM. Section 6.1 of the chapter discusses the technologies used for developing the AW agent package. Section 6.2 describes the overall design of the AW agent package. Section 6.3 describes the main components of the AW agent package. Section 6.4 describes the architecture of the AW agent package. Section 6.5 explains the basic functionality of the AW agent package. The chapter finishes in Section 6.6 with a brief discussion of two user guides that are provided for using the AW agent package to implement and extend the functionality of agents.

1.7.6. Demonstration of the Functionality of the VWADM

Chapter 7 presents three case studies that are used to demonstrate the functionality of the VWADM. Section 7.1 of the chapter shows the use of the VWADM to design an office application. Section 7.2 shows the use of the VWADM to design a living space application. Section 7.3 shows the use of the VWADM to design a student centre application. The chapter finishes in Section 7.4 with a review of the functionality of the VWADM in light of the three case studies.

1.7.7. Evaluation of the VWADM

Chapter 8 discusses a user study that was conducted as part of the evaluation of the VWADM. Section 8.1 provides an overview of the fitness-utility model, which was used to

design the study. Section 8.2 provides a brief overview of the methods and instruments used in the study. Section 8.3 reports on the preparatory activities that were performed to recruit developers to participate in the study. Section 8.4 discusses some ethical aspects about the study. Section 8.5 gives a summary of the procedure that was used to conduct the study. Section 8.6 provides an overview of participant experience information from the study. Section 8.7 reports on the reliability of the questionnaire used in the study. Section 8.8 presents the results of the study. The chapter finishes in Section 8.9 with a discussion of the results of the study.

1.7.8. Conclusion

Chapter 9 is the final chapter of the thesis. Section 9.1 of the chapter provides a summary of the thesis. Section 9.2 revisits the research hypothesis. Section 9.3 discusses some of the limitations of the research. Section 9.4 provides an overview of some of the benefits of the VWADM for developers. Section 9.5 discusses future work related to the research. Section 9.6 discusses the outlook for the research. The chapter finishes in Section 9.7 with some final remarks.

2. Literature Review

This chapter reviews some of the immediate areas that influenced the development of the research in terms of the state of the art in designing VW applications. Using an SE lens, the chapter firstly provides a definition of design. Next, it discusses a few of the reasons design is important, especially in the context of developing VW applications. This is followed by a review of some examples of desktop-based VR systems, which provide platforms that accommodate and facilitate designing VW applications. The chapter then provides a review of some of the existing methods in SE that are currently used for designing VW applications. This review is followed by a brief discussion of the architectural concept of placeness. The chapter finishes with a brief examination of place-oriented modelling techniques in architecture.

2.1. What is Design?

In SE, design can be defined as the specification of an object, which is manifested by an agent and is intended to accomplish certain goals in a particular environment, using a set of primitive components, while satisfying a set of requirements that are subject to constraints (Ralph & Wand, 2008). The object of design is the artefact that is being designed, which may be a thing or process. Otherwise, the agent is the subject of design - it is the human or computational entity that creates the specification of the object. All artificial things or processes are made from other things or processes called components (March & Smith, 1995), of which the lowest levels are in turn referred to as primitives (Devedzic, et al., 2000). As such, primitives are the basic components or building blocks that can be assembled or transformed by an agent to create an artefact.

The outcome of the design process is not always the artefact itself that is being designed. Instead, it may be a plan for its construction. As a result, design may sometimes be regarded as a planning process as opposed to a building process (McKay, et al., 2012). Nevertheless, the congruity between these two viewpoints is that the agent specifies properties of the artefact, which can be purely mental (El-Far & Whittaker, 2001), a symbolic representation (Tse & Pong, 1989), a physical model (Budde, et al., 1992) or a manifestation of the artefact itself (Garrett, et al., 2007). The properties of an artefact that an agent specifies are referred to as its specification. Essentially, the specification of an artefact is a description of its structural and/or behavioural properties (Roy, et al., 2001) such as the primitives that need to be assembled, how they are assembled or linked and the manner in which they are intended to function as a whole. Together, they form the requirements of the artefact, which are subject to constraints such as time, cost, scope and resources.

Design belongs to a class of teleological behaviours (Churchman, 1971; Franke, 1989), which are purposeful goal-seeking behaviours (Venkatasubramanian, 2007). As such, goals do not necessarily have to be explicit or well-defined. For example, some artefacts are designed simply based on the idea that they may be fun to use or based on widely perceived notions of usefulness or value. However, a survey of the literature on design suggests that while goals may not always be explicit or well-defined, the design process itself is always intentional (Rittel, 1987; Rosenman & Gero, 1998). On the contrary, unintentional artefacts are not designed. Therefore, goals are inherent in design, as long as the design is intentional. Design is usually situated in some sort of environment (Ralph & Wand, 2009). Such an environment may be termed as a context (Stauffer, et al., 1991), a setting (Twidale, et al., 1993) or a domain (Fischer, 1994). In addition, it may be characterised as an office, a studio or a computer environment. Finally, design usually occurs in two different types of environments, which are the environment in which the agent is situated and that in which the artefact is situated. Nevertheless, it is possible for both the agent and the artefact to share the same environment as is the case in this piece of work.

2.2. Why Design VW Applications?

There are a number of reasons why designing VW applications is important. From an architectural point of view, it would be unexpected for places in the real world such as houses, churches, supermarkets or any other engineered artefact for that matter (e.g., cars, computers and televisions) to be built without first designing them and producing plans for their construction. VW applications are the same: constructing them without firstly having some plans is hardly likely to produce good results (Yan & Damian, 2008). Nonetheless, there are more practical reasons for designing VW applications, which have their basis in SE (Curtis, et al., 1988; Bell, et al., 1994) as follows:

- It helps us to transform various requirements for these applications into specifications for constructing them.
- It helps us to translate specifications of these applications into coherent architectural structures, which can be mapped to an implementation environment.
- It helps us to manage some of the complexities that are associated with the construction of these types of applications.

2.3. Platforms for Designing VW Applications

There are many desktop-based VR systems from different developers and vendors in existence today, which provide a platform and an environment for designing VW applications. Some of these systems are briefly reviewed as follows:

Active Worlds is one of the oldest of desktop-based VR systems that are around today. It was developed by Ron Britvich and first launched as AlphaWorld in 1995. Active worlds focuses on content creation and user communities. It allows people to assign themselves a unique name, log in to a VE and explore VWs, which include content created by others. In addition, people can chat with one another and build structures on private or publicly owned land, using a selection of objects. Active Worlds has a built-in object manipulation system, which users can utilise to build objects, copy them, specify their exact placement in all three dimensions and write simple scripts using RenderWare Script (RWX) to give them a texture.

It also has a built-in scripting system that enables users to create dynamic and interactive objects. Fig. 2.1 shows a VW rendered by Active Worlds.

Figure 2.1. A VW rendered by Active Worlds

Second Life is one of the most popular of desktop-based VR systems in existence right now. It was developed by Linden Lab and first launched in 2003. Second Life focuses on content creation, entrepreneurship and user communities. It enables people to create avatars that can interact with other avatars, objects and places in VWs. It also enables people to explore its VWs, socialise, participate in individual and group activities and build, shop and trade virtual property and services with one another. Second Life has built-in 3D modelling tools and a collection of simple geometric shapes, which enable people to build virtual objects. In addition, it has a scripting system that uses a procedural scripting language called Linden Scripting Language (LSL), which can be used to add interactivity to objects. Fig. 2.2 shows a VW rendered by Second Life.

Figure 2.2. A VW rendered by Second Life

Open Simulator is a free and open-source desktop-based VR system. It was developed by Darren Guard and first launched in 2007. Open Simulator focuses on content creation, user communities and pushing the boundaries of VR systems through experimentation. For example, it uses an architecture called the hypergrid (Lopes, 2011), which allows users to be able to teleport between multiple Open Simulator-based VWs that do not necessarily have to be located on the same server. Similar to Active Worlds and Second Life, Open Simulator enables people to build structures on lands that grant the right to do so. As such, it has a built-in 3D modelling and scripting system, which users can utilise to create objects and subsequently use or trade them. The scripting system allows LSL or the C# programming language to be used to add interactivity to objects. Fig. 2.3 shows a VW rendered by Open Simulator.

Figure 2.3. A VW rendered by Open Simulator

High Fidelity is a free and open-source hybrid cloud-hosted desktop-based VR system for creating, hosting and exploring shared virtual experiences. It was developed by High Fidelity Incorporated and first launched in 2013. High Fidelity enables people to use common formats, tools and languages such as audio, facial recognition and scriptable application programming interfaces (APIs) to build complex interactive virtual experiences. In addition, it can scale to host thousands of people and petabytes of data in a single place and allow them to share processing power. High Fidelity has a marketplace for buying and selling content, built-in editing tools for content creation and a scripting system that uses the JavaScript language for adding interactivity to content. Fig. 2.4 shows a VW rendered by High Fidelity.

Figure 2.4. A VW rendered by High Fidelity

Meshmoon is a cloud-hosted desktop-based VR system that is based on realXtend technologies such as Webtundra and the Tundra 3D software development kit (SDK). It was developed by Admino Technologies and first launched around 2014. Meshmoon focuses on collaborative content creation and projects and pushing the boundaries of VR systems through experimentation and interoperability with other VR systems (e.g., Open Simulator) and non-VR systems such as social media platforms and the web. It allows people to create VWs and to script objects in them using the JavaScript language. Fig. 2.5 shows a VW rendered by Meshmoon.

Figure 2.5. A VW rendered by Meshmoon

Sansar is a new desktop-based VR system for creating and sharing virtual experiences. It is being developed by Linden Lab and is currently in the open beta phase for creators. Sansar enables scenes to be designed either in an immersive or non-immersive mode and to enhance them with interactive lighting, spatial audio and C# scripting. Third-party tools such as 3ds Max, Maya and Blender can be used to create 3D models, which can then be imported into Sansar as .fbx files. Linden Lab provides a store for Sansar, from which additional assets can be obtained for use in scenes. A scene in Sansar includes detailed avatars with speech-driven facial animations and motion-driven body animations that offer users a high quality virtual experience as they interact with creations. A key feature that distinguishes Sansar from other desktop-based VR systems is that a virtual experience (i.e., a scene and all of its content that a user can experience and with which they can interact) can be shared with others. As such, Sansar is a highly social desktop-based VR system, which provides a platform that enables users to have many virtual experiences and for them to be able to share their own with others. Fig. 2.6 shows a VW rendered by Sansar.

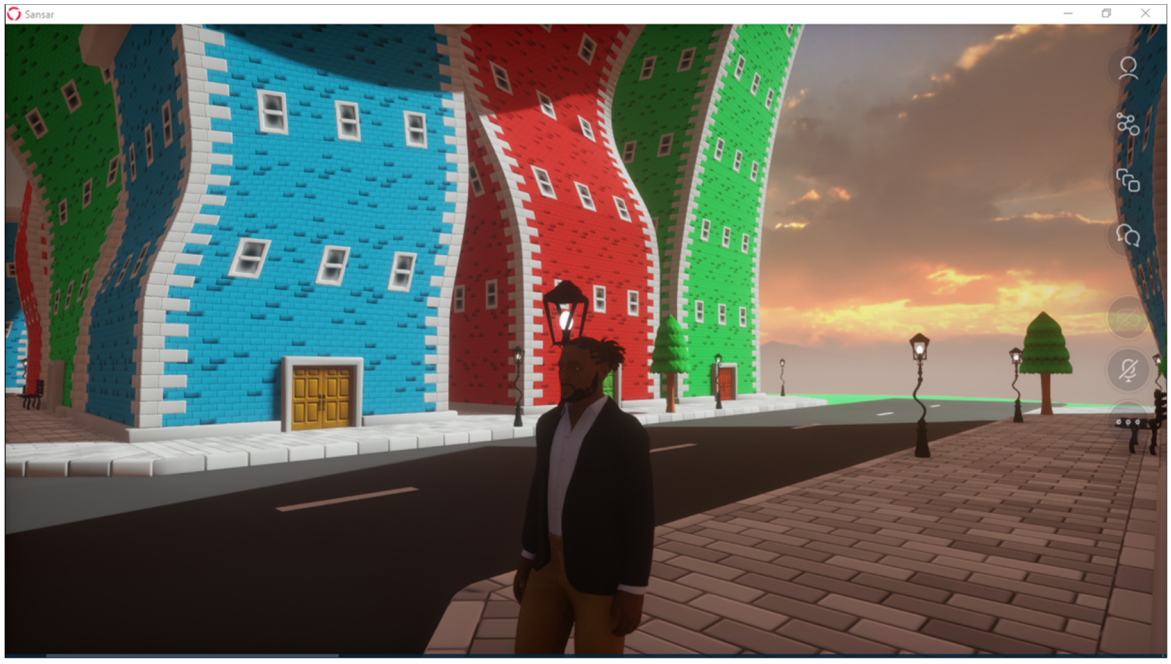


Figure 2.6. A VW rendered by Sansar

The various desktop-based VR systems offer a diverse set of tools and affordances, which can be used to develop VW applications. Nevertheless, these VW applications are characteristically similar to each other with regards to their structure and behaviour (Pellas, et al., 2017). This aspect enables a method for designing VW applications for a particular desktop-based VR system to be easily portable to another one of these systems.

2.4. Methods for Designing VW Applications

In the context of design in SE, a method is a way for solving certain classes of problems (Gregory, 1966), using some viewpoint such as an object-oriented, data-oriented or event-oriented viewpoint (Finkelstein, et al., 1990). It is usually expected to provide a series of steps or guidelines that indicate how to create an artefact, as well as a system for capturing semantic knowledge about the problems that it is meant to address and their solutions. As already mentioned, so far, developers have relied on the use of several existing methods in SE such as the structured, behavioural, formal, interactional and iterative methods for creating models of VW applications during the design process. They are reviewed as follows:

Structured methods are those that focus on providing a way for determining the components that comprise VW applications and how they link (i.e., their relationship) to one another (Dobrica & Niemela, 2002). They are usually based on an object-oriented viewpoint and involve the creation of conceptual models such as class (Oliveira, et al., 2003; El-Khalili, 2009) and component (Cheng & Yeh, 2007; Sawyerr & Pinkwart, 2011; Sawyerr & Pinkwart, 2011) diagrams, which are used for designing VW applications.

Behavioural methods are those that focus on providing a way for describing or representing the dynamic behaviour of VW applications as they are executing (Shaw, 1995). They are usually based on data-oriented and event-oriented viewpoints and involve the creation of conceptual models such as activity (Chevaillier, et al., 2012; Carvalho, et al., 2015; Madni, 2018) and state (Green, 1995; Polcar, et al., 2016) diagrams, which are used for designing VW applications.

Formal methods are those that focus on providing a way for creating and verifying the specifications of VW applications and the components that comprise them (Clarke & Wing, 1996). They are usually based on mathematical viewpoints and involve the creation of conceptual models such as algebraic (Schooten, et al., 1999; Zuniga, et al., 2005) and other model-based or formulaic (Zhou, et al., 2000; Figueroa, et al., 2008; Tarkan, 2009) specifications, which are used for designing VW applications.

Interactional methods are those that focus on providing a way for determining and describing interactions that involve VW applications such as their user inputs and outputs, interactions between these applications and other applications or systems and interactions between the components that comprise all of them (Moggridge, 2007; Sharp, et al., 2007). They are usually based on interaction-oriented viewpoints and involve the creation of conceptual models such as use case (Molina, et al., 2003; Reis, et al., 2012) or sequence (Winterbottom, et al., 2005; Tizani, 2011) diagrams, which are used for designing VW applications.

Iterative methods are those that focus on providing a way for prototyping, testing, analysing and refining the specifications of VW applications (Nielsen, 1993). They are usually based on a cyclic process and involve the creation of conceptual models such as wireframes (Roussou, et al., 2004; Srinivasan, et al., 2005), mockups (Whisker, et al., 2003; Billinghurst, et al., 2005; Maldovan, et al., 2006) and usability or user experience guidelines (Kaur, 1997; Gabbard, et al., 1999; Bowman, et al., 2001; Minocha & Reeves, 2010), which are used for designing specific parts of these applications (e.g., UIs, middleware or data persistence mechanisms).

2.5. Placeness

Users of VWs engage in a wide range of in-world activities such as learning and training (Ritsos, et al., 2013; Savin-Baden, et al., 2017), gaming (Bainbridge, 2014), socialising (Maentymaeki & Riemer, 2014) and attending meetings and events (Dodgson, et al., 2013). These activities are performed in and around VW applications, which results in the notion of placeness being imposed on them. If placeness is the consequence of the activities and conceptions of the inhabitants of a space, then place refers to the physical attributes that frame those activities and provide a socially shareable setting for them in terms of cues that organise and direct appropriate social behaviour in that particular place (Kalay & Marx, 2001). Based on arguments such as this, VW applications can be viewed as places that frame the in-world activities of users of VWs and provide them with a social setting for those activities.

2.6. Place-Oriented Modelling Techniques

The place-oriented viewpoint introduces a new approach for designing VW applications, which puts special requirements on existing methods in SE in terms of case-specific knowledge such as patterns, conventions and domain experience. As such, the research drew from the following place-oriented modelling techniques in architecture to help address the issue:

Architectural drawing is the practice of designing places using drawings (Smith, 2012). It involves creating the layout and visual appearance (or boundary) of buildings using sketches and diagrams. In architectural drawing, an architect uses drawing as a tool for solving spatial problems by recursively adding, subtracting, transforming and rearranging elements and materials until reaching a solution (Schaller, 1997). As such, sketches and diagrams employ a range of graphical indicators such as shapes, size and position, to explore and communicate ideas about such problems and their solutions.

Generative design is an established method in architecture that uses rules to create objects that have different shapes (Sedrez & Periera, 2012). It enables architects to be able to generate physical models of places, which they can review prior to construction. In as much as generative design is well-known for being used for architectural design in the real world (Flemming, 1995; Gero, 1996; Caldas & Rocha, 2001), it has also been used widely for similar purposes in VWs (Chien & Flemming, 1997; Gao & Gu, 2009; Steino, 2010; Singh & Gu, 2012). While the development of a generative design system or mechanism requires a certain level of technical skills, its use or application is comparatively much easier.

The VWADM uses architectural drawing to establish the means for effectively capturing, storing, describing and presenting spatial data about VW applications. In using this technique, architects do not build actual places. Instead, they create the instructions (usually in the form of plans) that enable others to be able to do so. As such, in order to manage the complexity of the process, the VWADM uses generative design (by proxy of a design agent with generative capabilities) to establish the means for interpreting and executing such instructions and generating physical models based on them.

2.7. Summary

This chapter reviewed some of the immediate areas that influenced the development of the research. As such, it provided a working definition of design in the context of SE. It also

gave a few reasons as to why design is important, especially for developing VW applications. A review was given of some examples of desktop-based VR systems, which provide platforms that accommodate and facilitate designing VW applications. In addition, a review was given of some of the existing methods in SE that are currently used for designing VW applications. The chapter provided a brief discussion of the architectural concept of placeness. Finally, a review was provided of some place-oriented modelling techniques in architecture.

3. Methodology

This chapter presents the design science research (DSR) paradigm, which was used to conduct the research, including the development of the VWADM. The chapter first provides an overview of DSR. It then discusses the rationale for using DSR. This is followed by an overview of the core principles of DSR. Next, a discussion is given about the DSR process model that was used for conducting the research. The chapter finishes with the review of a case study that was used as a pilot study for the research.

3.1. What is DSR?

DSR is a problem-solving paradigm that provides a set of synthetic and analytical techniques and perspectives for performing sociotechnical research (Hevner, et al., 2004). It involves the development of novel or innovative artefacts and analysis of their use and/or performance in order to understand and improve the behaviour of aspects of technology. Artefacts may be theories, frameworks, constructs, models, methods or instantiations.

3.2. Rationale for using DSR

Several other methodologies were examined early in the research for their potential use for developing the VWADM. However, since methodologies are usually discipline or activity-specific[4], they could not be used effectively for developing the VWADM. For example, methodologies such as Agile (Shore & Warden, 2008) and Spiral (Boehm, 2000) are used to manage software development. Others such as Lean (Poppendieck & Poppendieck,

[4] See (Huppatz, 2015) for his notes on Simon Herbert's Science of Design or (Diesing, 2017) Patterns of Discovery in the Social Sciences.

2003) are used to manage the processes that surround it. Design-centric methodologies such as design-based research (Brown, 1992; Mor, 2010) and critical artefact methodology (Bowen, 2009) are used for developing software and hardware artefacts (including non-computing products), but generally not to the extent that includes intangibles such as methods. Consequently, DSR was chosen to conduct the research because it supports the development of methods (Sawyerr, 2016). Furthermore, DSR offers strong support for the development process and provides several frameworks for understanding, conducting, evaluating and reporting research (e.g., the organisation of this thesis). Most of all, DSR was chosen because it is a well-established methodology that has been used for decades in SE for developing new algorithms, new compilers, new programming languages and new data models (Livari, 2007). As such, it is time-tested and has a wide knowledgebase to support research and development of a wide range of different types of artefacts.

3.3. Core Principles of DSR

The fundamental principle of DSR is that knowledge and understanding of a design problem and its solution are acquired through the building and evaluation of artefacts. Based on this principle, researchers apply artefacts to an application context for the purpose of addressing the problems (or needs) of that context. The usefulness of artefacts and the characteristics of the application context such as its people, technologies and development methodologies determine the extent to which this purpose is achieved. Ultimately, researchers use DSR to produce new artefacts to improve the ability of the application context to adapt and succeed in the presence of changing environments (Gregor & Hevner, 2013). Such artefacts embody the ideas, practices, technical capabilities and products through which technology can be effectively developed and efficiently used. Fig. 3.1 shows the DSR framework, which depicts these principles.

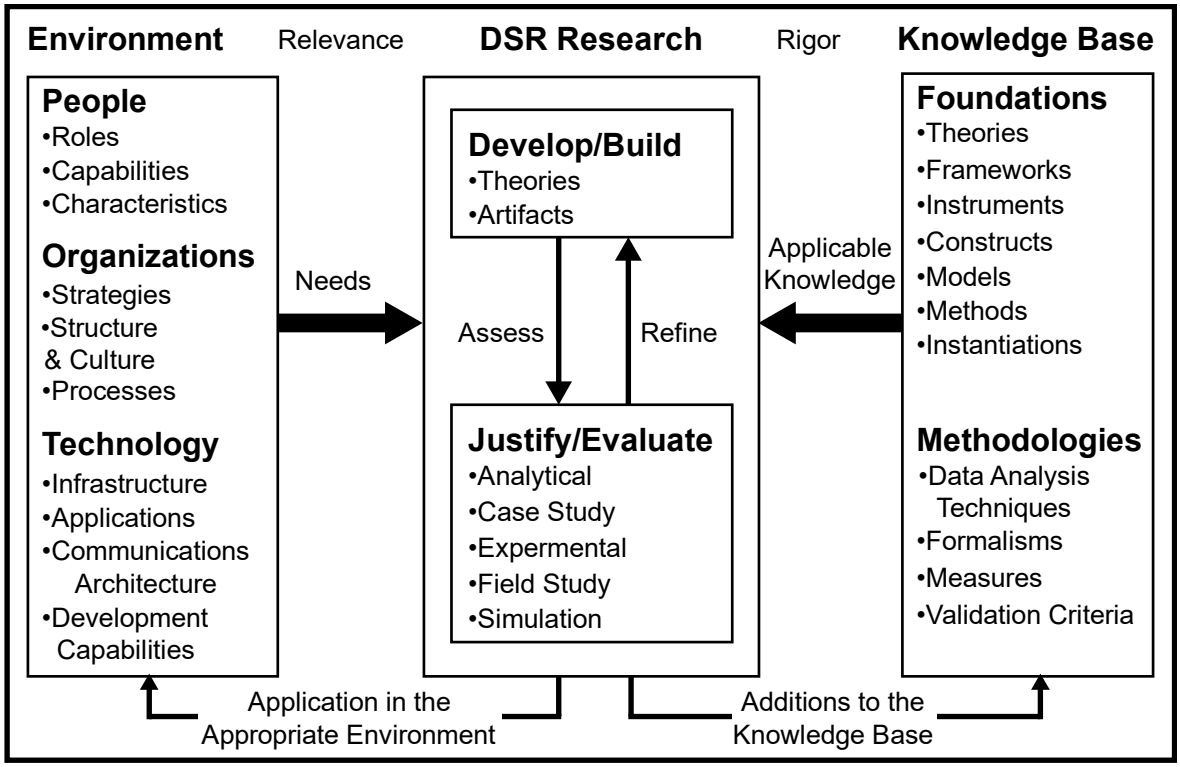


Figure 3.1. DSR framework (Hevner, et al., 2004)

The application context (or environment) defines the problem space in which the phenomena of interest resides. It is usually comprised of groups of people, businesses (or organisations) and their existing or planned technologies (Silver, et al., 1995). In it can be found the goals, tasks, problems and opportunities that define the needs of the environment as they are perceived by people (including those who are part of organisations). Such perceptions are shaped by the roles, capabilities and characteristics of people in their groups and organisations. Organisational needs are assessed and evaluated in the context of strategies, structure, culture and existing business processes. People and organisations are positioned relatively to existing or planned technological infrastructure, applications, communications architecture and development capabilities. Together, all of these (i.e., people, organisations and technology) define the problem or need with which the researcher engages.

3.4. DSR Process for Developing the VWADM

In this research, Peffers' DSR process model (Peffers, et al., 2008) was used as a guide to develop the VWADM. The Peffers' DSR process model is a research process model, which

is the embodiment of the science of the artificial (Simon, 1996), or what is also known as the science of design (Cross, 2001; March & Storey, 2008). As such, the Peffers DSR process model is a guide for the application of the scientific method to the task of discovering answers (i.e., solutions) to questions (i.e., problems) in DSR.

There are a number of other existing DSR process models such as those suggested by (Archer, 1984), (Takeda, et al., 1990), (Eekels & Roozenburg, 1991), (Walls, et al., 1992), which could have been used for conducting the research and especially for developing the VWADM. However, the Peffers DSR process model was chosen because it is the most mature of the models, which is generally accepted for conducting DSR. The Peffers DSR process model is consistent with prior literature relating to DSR processes (and process models). In addition, it provides a nominal process model for conducting DSR and a mental model for presenting and reviewing it. Fig. 3.2 shows the conceptual model of the DSR process (i.e., the Peffers DSR process model) that was used for developing the VWADM.

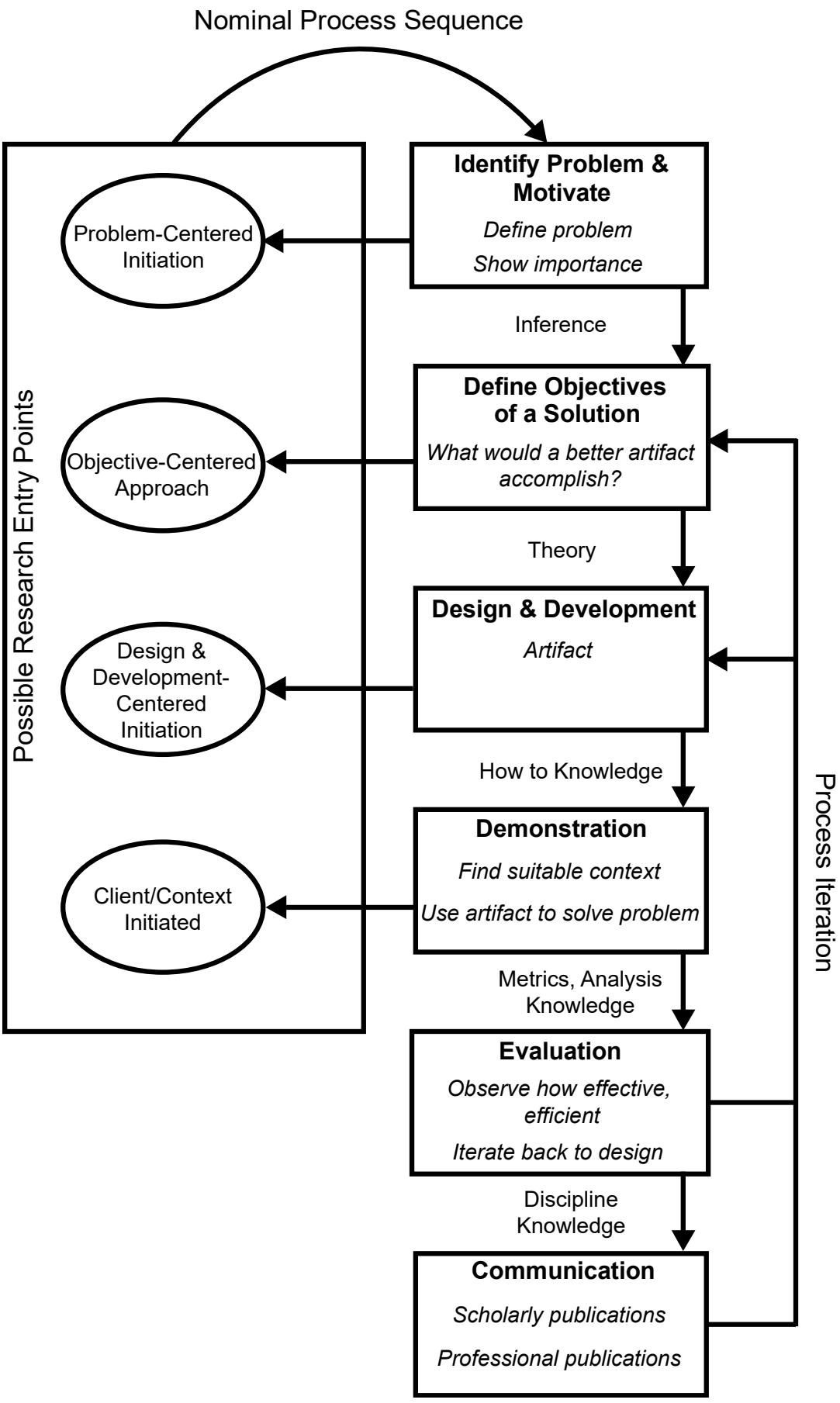


Figure 3.2. A conceptual model of the DSR process for developing the VWADM (Peffers, et al., 2008)

Using Peffers' DSR process model, the VWADM was developed using the following six steps:

3.4.1. Identify Problem & Motivate

The first step in developing the VWADM was to define the research problem and motivation for the work (see Section 1.1 in Chapter 1). As such, the research began by identifying the need to provide support for the place-oriented viewpoint when designing VW applications and explored place design in architecture for ways to address the issue. Literature reviews were conducted to gain an understanding of the state of the art in designing VW applications, as well as the importance of a solution amongst other things (see Chapter 2). This stage of the research also included the mining of information in-world and from online forums dedicated to desktop VR systems about design problems and how they affect the people that use these platforms.

3.4.2. Define Objectives of a Solution

The second step in developing the VWADM was to infer the objectives of a solution by extrapolating them from the problem definition and data gathered about what is possible and feasible in VWs and when designing VW applications (see Section 1.3 in Chapter 1). During extrapolation, knowledge about the state of problems and current solutions was used with emphasis being put on the manner in which the VWADM is expected to support designing VW applications.

3.4.3. Design & Development

The third step in developing the VWADM was to determine its composition and desired functionality using theory and design and development knowledge related to linguistic design in VWs, worldviews for designing in VWs and agent-based design in VWs, which was brought to bear in the new method. Chapter 4 elaborates on the theory and design and development knowledge that was used to develop the VWADM. In addition, Chapter 5 provides a detailed description of the composition and functionality of the VWADM.

Furthermore, Chapter 6 provides an overview of an implementation of the VWADM's mechanism for designing VW applications.

3.4.4. Demonstration

The fourth step in developing the VWADM was to demonstrate its functionality by applying it to a few design scenarios. Chapter 7 is about such a demonstration.

3.4.5. Evaluation

The fifth step in developing the VWADM was to conduct a user study to evaluate its features and capabilities with respect to their significance for practical application in designing VW applications. Such a study involved obtaining the opinions of users (developers) about the VWADM's fitness and utility for designing VW applications. Chapter 8 presents the evaluation.

3.4.6. Communication

The sixth and final step in developing the VWADM involves discussing the problem (and its motivation), describing the solution (i.e., the design and development of the VWADM), evaluating it against some criteria of interest and reporting on all of these things (for the benefit of various stakeholders of the research such as developers, practitioners and other researchers). This thesis is an example of such communication.

The first and second steps of the Peffers DSR model (identify problem & motivate and define objectives of a solution) belong to the problem identification phase of the approach used to conduct this research. The third step (design & development) belongs to the solution design phase. The fourth and fifth steps (demonstration and evaluation) belong to the evaluation phase. All of these steps lead to the communication of a summary of the results.

3.5. Design and Development Iteration

DSR considers design and development to be an iterative search process (Venable, 2010; Marheineke, 2016). As such, the Peffers DSR model supports process iteration, which is reflected in research that was conducted on methods for designing VW applications prior to arriving at the VWADM (Sawyerr, et al., 2013; Sawyerr & Hobbs, 2014). The summary of a case study that was used as a pilot study for the research is given as follows:

The purpose for conducting the case study was to determine the effectiveness of using an existing method for designing VW applications. The example VW application used in the case study was the Chaotic Science Lab (Holley, et al., 2013), a VW application that aims to provide support for trainee teachers in developing health and safety skills. Fig. 3.3 shows the Chaotic Lab VW application.

Figure 3.3. The Chaotic Lab VW application

The case study used techniques from iterative design and the associated tools were an extended cognitive walkthrough (see Appendix A) and VW heuristics (see Appendix B). Iterative design involves the cyclic refinement of a product's design based mainly on testing (Kruchten, 2000). The extended cognitive walkthrough is an adaption of the classic cognitive walkthrough, which is a task-based inspection method for conducting usability evaluations (Wharton, et al., 1994). The VW heuristics is an adaption of the classic heuristics (Nielsen & Molich, 1990), which is a set of guidelines for evaluating a GUI and for identifying design problems in it.

The iterative design method was chosen because of its known rigour in finding and eliminating design problems, which is based on a cycle of testing and (re)design. The extended cognitive walkthrough and the VW heuristics were chosen because they were developed to specifically address design issues that pertain to VWs. Furthermore, both of them complement the iterative design method. The extended cognitive walkthrough and the VW heuristics were both used in a two-stage procedure in the case study.

The first stage of the case study involved a scenario to prepare a virtual classroom that was scattered with various science apparatus and hazards (e.g., flasks containing chemicals) and health and safety equipment (e.g., fire extinguishers) for a new teaching session (assuming a recently ended teaching session), while following basic health and safety procedures. The user was given 10 minutes to put all the items that were used during the previous teaching session back into their correct places. At the end of the task, the Chaotic Lab VW application provided the user with their score. Further details about the user's current and previous sessions such as times, dates, locations and score history were accessible via a web-based interface.

The extended cognitive walkthrough was used to evaluate three modes of interaction: task action, navigation and system initiative. As such, in the previously described scenario, the user forms an intention to achieve a task goal and proceeds to navigate around the Chaotic Lab VW application with some objective in mind. System initiative may occur during navigation. During system initiative, the task action and navigation are interrupted in order to provide some guidance or help. The user has the option of engaging with system initiative or declining it, after which task action and navigation is resumed.

The findings from the first stage of the case study suggest that the extended cognitive walkthrough is a reactive tool that is applied late in the development process. As such, it belongs to the testing phase of the software development process as opposed to its design phase. Furthermore, the findings from this stage of the case study suggest that the extended

cognitive walkthrough would be more useful for identifying task-based design issues in VW applications with reference to the activities that they are intended to support.

The second stage of the case study involved the use of VW heuristics to identify design issues in the GUI of the Chaotic Lab VW application. The VW heuristics included a checklist of 53 items (H1.1 - H16.3) grouped into 16 specific usability heuristics (H1 - H16) and further grouped into three categories as follows: (1) design and aesthetics, (2) control and navigation and (3) errors and help. Table 3.1 shows a summary of the results of the VW heuristics evaluation.

Table 3.1. A summary of the results of the VW heuristics evaluation

Categories	Problems Detected	Mean Frequency	Mean Severity	Mean Criticality
Design and Aesthetics	18	0.50	1.25	1.75
Control and Navigation	12	1.10	1.48	2.58
Errors and Help	6	1.25	1.88	3.13
Total	36	2.85	4.61	7.46

The VW heuristics evaluation was conducted by three expert evaluators. Each evaluator separately used the VW heuristics checklist of 53 items to identify design issues in the GUI of the Chaotic Lab VW application. The problems that each evaluator found were listed (again separately) and grouped according to the 16 specific usability heuristics (H1 - H16). Each evaluator then gave a frequency and severity score to each of the 16 specific usability heuristics (H1 - H16) that ranged from 0 to 4 (0 means infrequent or negligible and 4 means frequent or severe). The set of all three scores were put together for each evaluator, resulting in three columns of scores (one for each evaluator) for frequency and also for severity. The arithmetic mean of the evaluators' scores was taken for each of the 16 specific usability heuristics (H1 - H16). In addition, the arithmetic mean was taken for each of the three categories design and aesthetics, control and navigation and errors and help to determine both the mean frequency and mean severity. Finally, the mean criticality was

determined by adding the mean severity and mean frequency scores together for each of the three categories design and aesthetics, control and navigation and errors and help.

The findings from the second stage of the case study also suggest that VW heuristics are a reactive tool that is applied late in the software development process. Therefore, as with the cognitive walkthrough, it belongs to the testing phase of the software development process as opposed to its design phase. In addition, the findings from this stage of the case study suggest that VW heuristics would be more useful for discovering problems in the GUI of VR systems rather than VW applications.

Overall, the findings from the case study suggest that both the extended cognitive walkthrough and VW heuristics are concerned with task-based analysis and usability evaluations respectively. These findings agree with the literature on the purpose of both of these methods (Nagpal, et al., 2017; Oliveira, et al., 2017). Nonetheless, more importantly, the overall findings led to the decision to explore other means (methods) to support the place-oriented viewpoint for designing VW applications.

3.6. Summary

This chapter presented the DSR paradigm, which was used to conduct the research. The chapter first provided an overview of DSR. It then discussed the rationale for using DSR. This was followed by an overview of the core principles of DSR. Next, a discussion was given about the DSR process model that was used for conducting the research. Finally, a review was given of a case study that was used as a pilot for the research.

4. Design and Development of the VWADM

This chapter discusses the design and development of the VWADM using theory and design and development knowledge related to linguistic design in VWs, worldviews for designing in VWs and agent-based design in VWs. First, linguistic design in VWs is discussed. This is followed by a discussion of views for designing in VWs. The chapter finishes with a discussion of agent-based design in VWs.

4.1. Linguistic Design in VWs

This section discusses theory and design and development knowledge that pertains to linguistic design in VWs, which covers the linguistic characterisation of VWs, grammar as a tool for designing VW applications and shape grammar.

4.1.1. Linguistic Characterisation of VWs

The modern concept of space extends beyond its originally strict geometrical meaning of an empty area to include matter characteristics (Nerlich, 1994). Spatial theory distinguishes between three types of space (Soja, 1985; Lefebvre, 1991), which are as follows:

- Physical space - comprised of nature and the cosmos
- Mental space - comprised of logical and formal abstractions
- Social space - comprised of social interactions

Each of these types of space is interlinked with the other and together they define the matter of space (Merrifield, 1993). For example, the space of nature and the cosmos cannot be considered separately from that of logical and formal abstractions and social interactions.

Based on this idea of spatial interlinks, it can also be argued that cyberspace cannot be considered separately from physical, mental and social space.

Parallels can be drawn between the three types of space mentioned above and Popper's three worlds (Popper, 1979), which is a way of looking at reality as defined in the following:

- World 1: comprised of physical objects and events, including biological entities.
- World 2: comprised of mental objects and events including dreams, thoughts and consciousness.
- World 3 comprised of objective knowledge or the products of thought, including abstract objects such as art, engineering and scientific theories.

For Popper, the three types of worlds are not separate either. Instead, they are interlinked and always interacting with each other. For example, events in World 1 and World 2 can cause changes in World 3, such as the development of scientific theories. Therefore, it can also be argued here that cyberspace respects the trichotomy of reality as is evident in similar arguments made in other areas of discourse such as sociology (Esping-Andersen, 1990; Weeden & Grusky, 2012), mathematics (Tall, 2004) and political science (Jahn, 2014). Fig. 4.1 shows the relationship between the three types of space and Popper's three worlds.

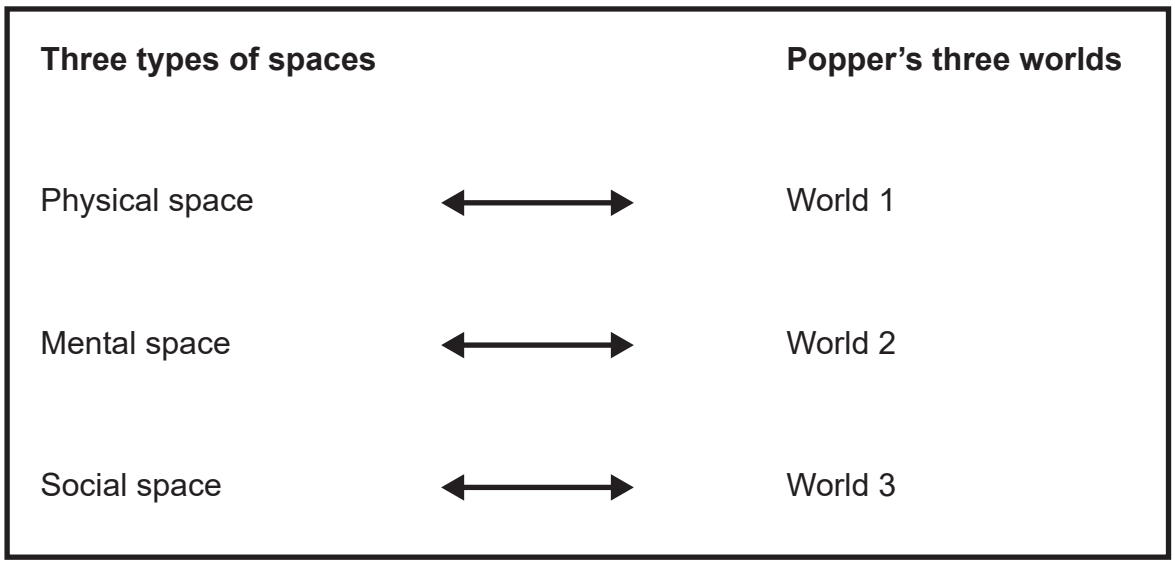


Figure 4.1. Relationship between the three types of space and Popper's three worlds

(Cicognani, 1998) defines cyberspace as an electronic flux of information, which means that it does not only rely on a computer-generated environment, but it also deals with speed, access and manipulation. In addition, she argues that the information that characterises cyberspace is structured on language. As such, she concludes that cyberspace is a linguistic construction, since any object found in it is a result of some sort of language such as the hypertext mark-up language (HTML), JavaScript or Java. VWs are part of cyberspace (Janelle & Hodge, 2000). Therefore, it can be concluded that they are interlinked with the three notions of space (i.e., physical, mental and social spaces) as well as Popper's three perspectives of reality (i.e., World 1, World 2 and World 3). More importantly, as objects in cyberspace, a final argument can be made that VWs are also characterised by (and structured on) language.

4.1.2. Grammar as a Tool for Designing VW Applications

Given that VWs are fundamentally linguistic in nature (Cicognani, 1998), grammar can be applied to the design process for VW applications (Cicognani, 2000). In formal language theory, a grammar is a set of production rules for strings in a formal language (Martinez, 2016). Such rules, describe how to form strings from the language's alphabet, which are valid according to the language's syntax. Consequently, a grammar is comprised of a set of rules for rewriting or transforming strings, as well as a start symbol from which such a process begins. In order to generate a new string, rules are applied recursively to a string consisting of a single start symbol until a string is produced that contains neither the start symbol nor designated nonterminal symbols. For example, given an alphabet that consists of x and y and a start symbol S, it is possible to derive two rules S → xSy (rule 1) and S → yx (rule 2) such that:

if rule 1 is applied, then string xSy is obtained;

if rule 1 is applied again, then xSy replaces S and string xxSyy is obtained;

if rule 2 is applied at this point, then yx replaces S and string xxyxyy is obtained.

The example can be simplified as follows:

$$S \Rightarrow xSy \Rightarrow xxSyy \Rightarrow xxyxyy$$

The concept of grammar as a tool for designing VW applications has its basis in speech acts (Austin, 1962), which are utterances that have performative functions. A speech act is the action performed by language to modify the state of an object on which the action is performed (Cicognani & Maher, 1997). Some examples of speech acts are as follows:

"I promise I will do my best"

"I baptise you George…"

"I now pronounce you husband and wife"

Utterances may be questions, convictions or actualities (i.e., facts, truths, proofs, etc.). However, these do not constitute speech acts. Instead, speech acts are those that are pronounced using the simple present tense indicative in order to affect an actual situation. As such, they are neither interrogative nor descriptive and they do not refer to past events.

4.1.3. Shape Grammar

The GDG framework is based on shape grammars (Stiny & Gips, 1972), which use a set of shape rules that can be applied to shape primitives in order to generate a set or language of designs (Knight, 2000). Shape rules have generative characteristics, as well as descriptive ones. Therefore, they can be used to capture and store data about some design problem as well as to generate a view that is the descriptive solution to such a problem. According to (Gips, 1975), shape grammars consist of four basic components as follows:

1. S: a finite set of shapes
2. L: a finite set of symbols
3. R: a finite set of shape rules
4. I: an initial shape

The shapes in the set S and the symbols in the set L provide the building blocks for defining the shape rules in set R and the initial shape I. Each shape rule follows the form of $S_a \rightarrow S_b$, where S_a and S_b are two different labelled shapes.

Shape grammars can be used to generate a shape-oriented language of design in a process that involves the recognition of a particular shape and its possible replacement. Shape rules specify the particular shapes to be replaced and the manner in which they are replaced. As such, underlying a set of shape rules are transformations, which permit one shape to be replaced by another.

A shape-oriented language of design can be generated by starting with an initial shape and applying shape rules to it in a stepwise manner. The result of this process is another shape consisting of the initial shape with the right-hand side of the rule replacing the left-hand side of the rule. According to (Stiny, 1980), the application of shape rules is as follows:

1. Find part of the shape that is geometrically similar to the left side of a rule in terms of both terminal and non-terminal elements.
2. Find the geometric transformations (scale, translation, rotation, mirror image) that make the left side of the rule identical to the corresponding part in the shape.
3. Apply those transformations to the right side of the rule.
4. Substitute the transformed right side of the rule for the part of the shape that corresponds to the left side of the rule.
5. The generation process is terminated when no rule in the grammar can be applied.

By alternating the sequence in which the above shape rules are applied, different designs that share a similar style can be generated. However, a point of interest when using shape grammars is the accuracy of the designs with regards to the design goals (including the satisfaction of design requirements and constraints). According to (Knight, 1999), there are two approaches that can be used to address this issue. The first approach is to incorporate the requirements and the constraints into the shape rules so that the generated designs have a certain level of predictability in meeting the design goals. The second approach is to generate the designs without incorporating the requirements and constraints into the shape rules (the rules will nevertheless still be based on an interpretation of these) and to subsequently use a design agent (i.e., human or computational) to evaluate them and select

the ones that satisfy the requirements and constraints. Fortunately, shape grammars are flexible enough to manage either approach without much of a trade-off in performance (e.g., accuracy). Therefore, a final point to be made about shape grammars is that they are easily understandable and usable by people as well as adaptable for use in computer programs.

4.2. Views for Designing in VWs

This section discusses theory and design and development knowledge that pertains to views for designing in VWs. Here, views are representations of VWs and objects in them, based on the place-oriented viewpoint. As such, this section discusses the three-layered view of VWs, as well as the three-layered view of objects in them.

4.2.1. Three-Layered View of VWs

(Gero & Kannengiesser, 2004) describe a situated view of design that assumes a non-static environment that is comprised of three different interlinked environments (all interacting with one another) in which the design process takes place. Fig. 4.2 shows the situated view of design.

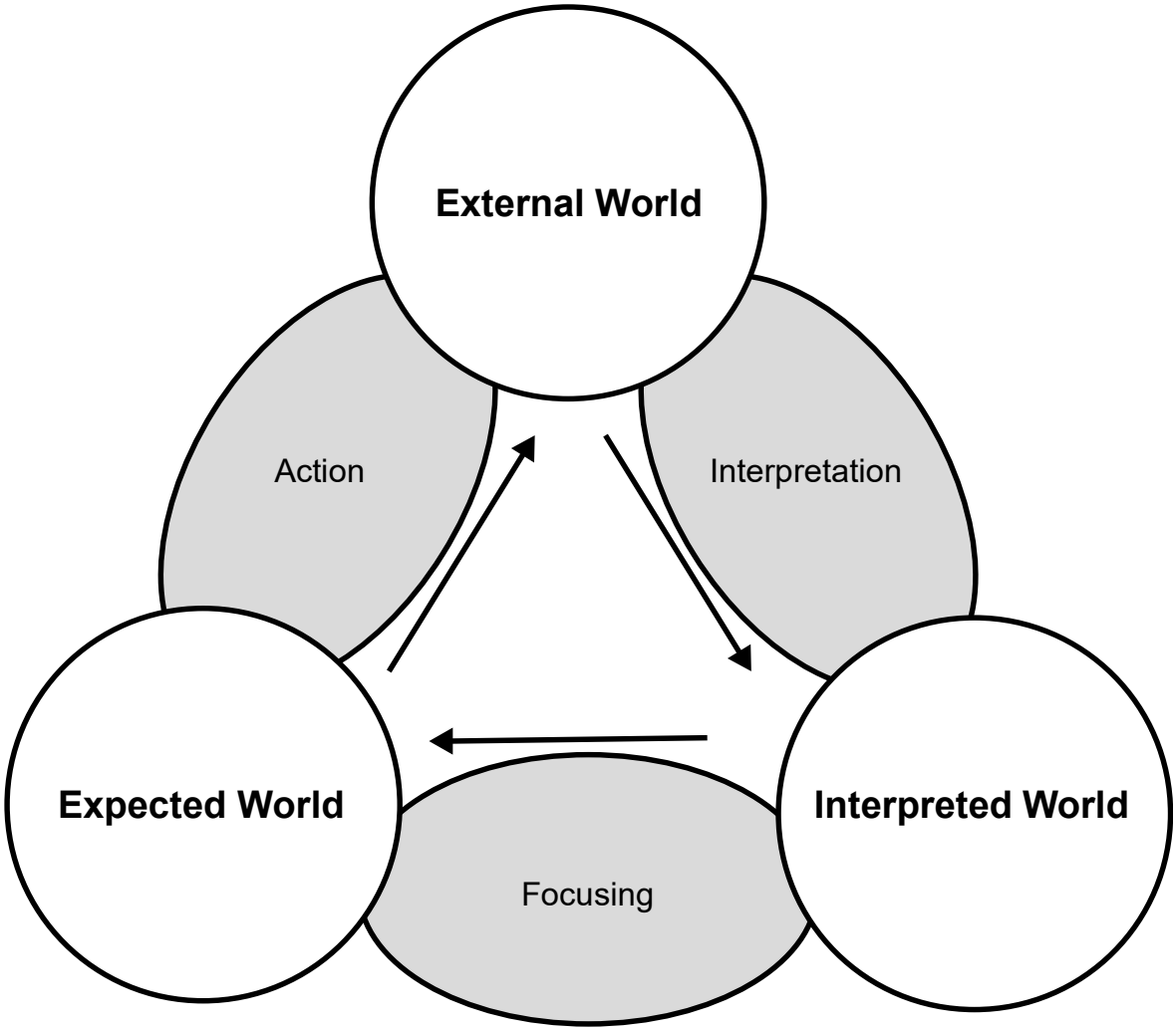


Figure 4.2. Situated view of design (Gero & Kannengiesser, 2004)

Based on this view, the GDA model's process for designing VW applications can be modelled as the interactions of three representation layers of a VW, which are the external world, the interpreted world and the expected world. The external world comprises representations outside a GDA. The interpreted world is the internal representation of the external world that exists inside the GDA, which reflects its knowledge and experience. The expected world is a part of the interpreted world in which the results of the design process are predicted based on the GDA's current design goals and its interpretations of the current state of the world.

The GDA model uses the three-layered view of VWs for capturing the spatial elements that comprise VW applications. For GDAs, a VW W is the union of the external world W_{ext}, the interpreted world W_{int}, and the expected world W_{exp} such that:

$$W = W_{ext} \cup W_{int} \cup W_{exp}$$

A VW is comprised of spatial elements, representations of its users in the form of playable characters or avatars and events that occur in the environment, all of which exist outside of the GDA. As such, a GDA's external world W_{ext} may include all or part of these things. The interpreted world W_{int} is the GDA's interpretation (and internal representation) of W_{ext}. The current design needs in the VW and the current state of the VW are represented in W_{int}. The expected world W_{exp} is where the GDA hypothesises about its design goals, generates the models of VW applications and initiates other changes in the VW. The GDA hypothesises by matching its interpretations of the current design needs against the current state of the VW, reads and applies the design specification from the GDG and generates a model of the VW application to satisfy the requirements.

The external world W_{ext} is comprised of A_{ext}, E_{ext}, wA_{ext} and O_{ext} such that:

$$W_{ext} = A_{ext} \cup E_{ext} \cup wA_{ext} \cup O_{ext}$$

where A_{ext}, E_{ext} and O_{ext} are each represented by a set of elements of the same type, and wA_{ext} is represented by an ordered list of properties such that:

$$A_{ext} = \{avatar_{ext\_1}, avatar_{ext\_2}, \ldots, avatar_{etx\_n}\}$$

$$E_{ext} = \{event_{ext\_1}, event_{ext\_2}, \ldots, event_{ext\_n}\}$$

$$wA_{ext} = \{size_{ext}, capacity_{ext}, owner_{ext}, server_{ext}, system\_time_{ext}\}$$

$$O_{ext} = \{O_{ext\_1}, O_{ext\_2}, \ldots, O_{ext\_n}\}$$

A_{ext} represents the avatars of users in the VW. E_{ext} represents the various events in the VW. wA_{ext} represents the properties of the VW. O_{ext} represents objects in the VW. In a VW, objects can refer to primitives that are constructed using other primitives (e.g., a chair or desk). As such, VW applications are also objects.

The interpreted world W_{int} is comprised of A_{int}, E_{int}, wA_{int} and O_{int} such that:

$$W_{int} = A_{int} \cup E_{int} \cup wA_{int} \cup O_{int}$$

where A_{int}, E_{int}, wA_{int} and O_{int} are the GDA's internal transformation of A_{ext}, E_{ext}, wA_{ext} and O_{ext} such that:

$$A_{int} = \tau(A_{ext})$$

$$E_{int} = \tau(E_{ext})$$

$$wA_{int} = \tau(wA_{ext})$$

$$O_{int} = \tau(O_{ext})$$

A_{int}, E_{int}, wA_{int} and O_{int} represent the GDA's interpretation (and internal representation) of A_{ext}, E_{ext}, wA_{ext} and O_{ext}. In addition, the GDA interprets the current design needs N in the VW and the current state of the VW sT such that:

$$N = \tau(W_{int})$$

$$sT \in W_{int}$$

The expected world W_{exp} is comprised of A_{exp}, E_{exp} and O_{exp} such that:

$$W_{exp} = A_{exp} \cup E_{exp} \cup O_{exp}$$

where A_{exp}, E_{exp} and O_{exp} represent the design goals (hypothesised by the GDA), which aim to eliminating mismatches between the current design needs N in the VW and the current state sT of the VW such that:

$$A_{exp} \in \tau A(N, sT)$$

$$E_{exp} \in \tau E(N, sT)$$

$$O_{exp} \in \tau O(N, sT)$$

A_{exp} represents the expected properties of the avatars of users in the VW. E_{exp} represents the expected events in the VW. O_{exp} represents the expected objects in the VW and their properties. In general, the properties of the VW are usually static. Therefore, the expected world W_{exp} does not include wA_{exp}.

4.2.2. Three-Layered View of Objects in VWs

In addition to a situated view of design, (Gero, 1990) introduced the function, structure and behaviour (FBS) model of design that generalises the design process using three variables:

function, behaviour and structure. The FBS model of design can be used to characterise elements of the design process in terms of the following sub-processes: formulation, synthesis, analysis, evaluation, production of design description (i.e., documentation) and reformulation. Fig. 4.3 shows the FBS model of design.

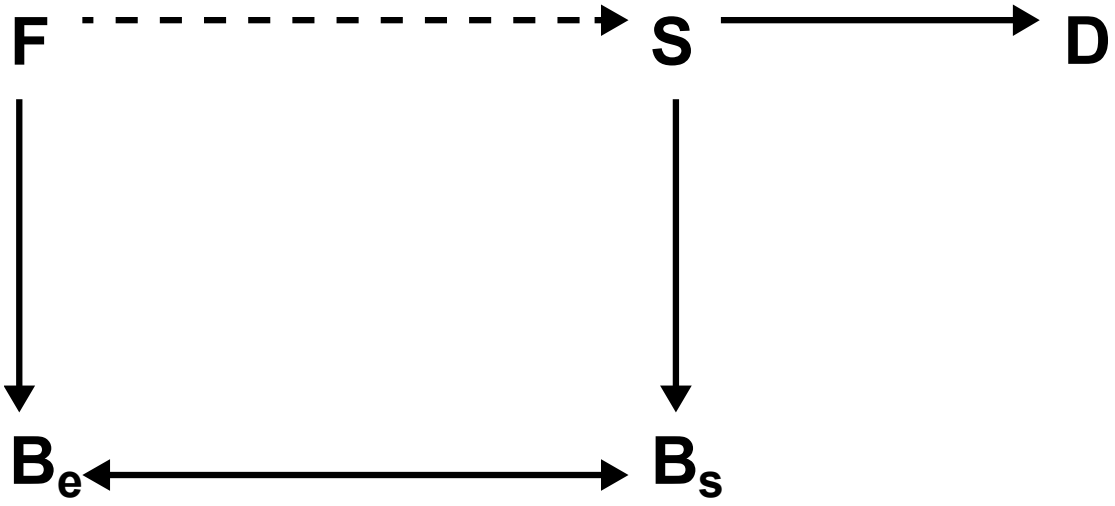


Figure 4.3. FBS model of design as a process (Gero, 1990)

The symbol F represents the set of functions, B_e represents the set of expected behaviours, S represents structure, B_s represents the set of actual behaviours and D represents design descriptions such that:

$$F + (B_e \Leftrightarrow B_a) + S \Rightarrow D$$

Sub-process flow based on the FBS model of design is as follows:

1. Formulation - the design process begins and the design requirements (i.e., functions) are transformed into the expected behaviours B_e.
2. Synthesis - the design solution is provided in the form of structures S, which are intended to support the expected behaviours B_e.
3. Analysis - the actual behaviours B_s are derived from the structures S.
4. Evaluation - the actual behaviours B_s are compared with the expected behaviours B_e.
5. Documentation - if the actual behaviours B_s are evaluated to be satisfactory, the documentation sub-process will be triggered to produce design descriptions D.

6. Reformulation - if the actual behaviours B_s are evaluated to be unsatisfactory, the reformulation sub-process will be triggered in order to adjust relevant elements and restart the flow.

An adapted version of the FBS model of design can be used to characterise the elements that are involved in the design process in VWs. Fig. 4.4 shows the FBS model of design, which is specifically for supporting the design process in VWs.

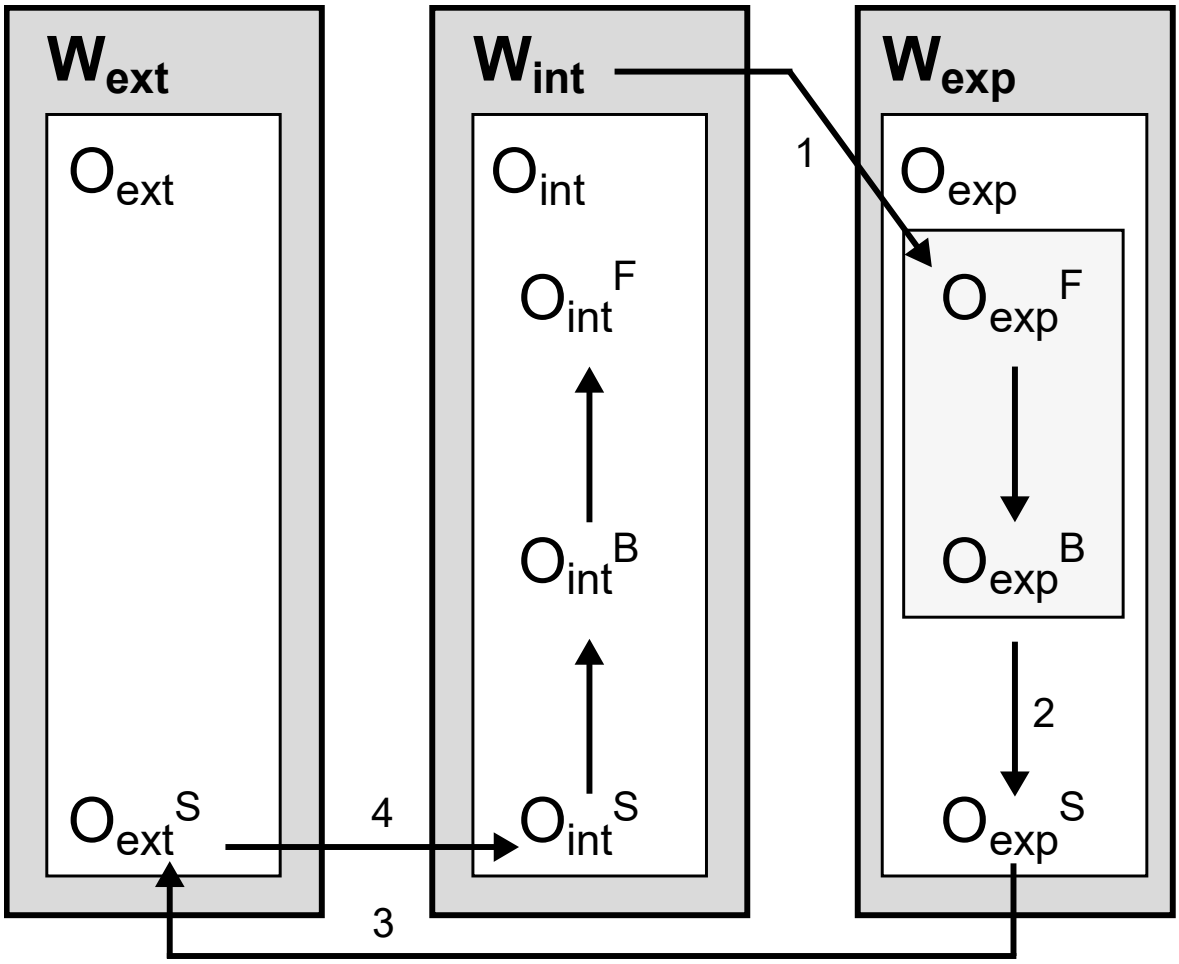


Figure 4.4. FBS model of design for VWs (Yu, et al., 2012)

The design process in VWs can be represented as interactions among W_{ext}, W_{int} and W_{exp}. As such, the FBS model of design for VWs can be used to characterise elements of the design process for VW applications in terms of the following sub-processes: (1) hypothesising, (2) designing, (3) action and (4) (re)interpretation. Sub-process flow based on the FBS model of design for VWs is as follows:

1. Hypothesising - the GDA sets up design goals in terms of the expected functions O_{exp}^{F} and the expected behaviours O_{exp}^{B}, in order to eliminate mismatches between the current design needs N in the VW and the current state sT of the VW such that:

$$O_{exp}^{F} = \{O_{exp\_1}^{F}, O_{exp\_2}^{F}, \ldots, O_{exp\_n}^{F}\}$$

$$O_{exp}^{B} = \{O_{exp\_1}^{B}, O_{exp\_2}^{B}, \ldots, O_{exp\_n}^{B}\}$$

for any expected function $O_{exp\_i}^{F}$ and expected behaviour $O_{exp\_i}^{B}$ such that:

$$O_{exp\_i}^{F} \in \tau(N, sT)$$

$$O_{exp\_i}^{B} = \tau(O_{exp\_i}^{F})$$

2. Designing - the GDA applies GDGs to satisfy its current design goals such that:

$$O_{exp}^{S} = \tau(O_{exp}^{F}, O_{exp}^{B})$$

3. Action - the actions for generating the model of a VW application are planned and activated in the VW such that:

$$O_{exp}^{S} \rightarrow O_{ext}^{S}$$

4. (Re) interpretation - generation of the model of the VW application causes further changes to the VW, which triggers (re)interpretation such that:

$$O_{int} = \tau(O_{ext})$$

for any interpreted object O_{int_i} in the VW such that:

$$O_{int\_i}^{S} = \tau(O_{ext\_i}^{S})$$

$$O_{int\_i}^{B} = \tau(O_{int\_i}^{S})$$

$$O_{int\_i}^{F} = \tau(O_{int\_i}^{B})$$

The GDA model uses the FBS model of design for VWs to represent objects in VWs. For the GDA model, any object O_i in the VW is comprised of functions O_i^F, behaviours O_i^B, and structures O_i^S such that:

$$O_i = O_i^F \cup O_i^B \cup O_i^S$$

The representation of an object in the external world includes structures $O_{ext\_i}{}^{S}$. However, functions $O_{ext\_i}{}^{F}$ and behaviors $O_{ext\_i}{}^{B}$ of the object are assumed to be nil, and are therefore addressed in the GDA's interpreted world such that:

$$O_{ext\_i}{}^{F} = \emptyset$$

$$O_{ext\_i}{}^{B} = \emptyset$$

$$O_{ext\_i}{}^{S} = \{structure_{ext\_1}, structure_{ext\_2}, ..., structure_{ext\_n}\}$$

All of the objects in the interpreted world have a counterpart in the external world. The interpreted functions $O_{int\_i}{}^{F}$ and behaviors $O_{int\_i}{}^{B}$ are derived from the interpreted structures $O_{int\_i}{}^{S}$ such that:

$$O_{int\_i}{}^{F} = \tau(O_{int\_i}{}^{B})$$

$$O_{int\_i}{}^{B} = \tau(O_{int\_i}{}^{S})$$

$$O_{int\_i}{}^{S} = \tau(O_{ext\_i}{}^{S})$$

It is optional for an object in the GDA's expected world to have a counterpart in the interpreted world, since objects in a VW can be generated based either on existing structures or by new creations. The GDA's design goals are hypothesised in terms of the expected functions $O_{exp\_i}{}^{F}$ and the expected behaviours $O_{exp\_i}{}^{B}$ and with regards to the current design needs N and to the current state sT of the VW such that:

$$O_{exp\_i}{}^{F} \in \tau(N, sT)$$

$$O_{exp\_i}{}^{B} = \tau(O_{exp\_i}{}^{F})$$

After a design goal has been hypothesised by the GDA, it reads and applies the design specification from the GDG and generates structures of the object (i.e., the VW application) represented by $O_{exp\_i}{}^{S}$, which are instantiated in the VW such that:

$$O_{exp\_i}{}^{S} = \tau(O_{exp\_i}{}^{F}, O_{exp\_i}{}^{B})$$

4.3. Agent-based Design in VWs

This section discusses theory and design and development knowledge that pertains to agent-based design in VWs. Agents are the basis for the intelligent component of the

VWADM. As such, this chapter discusses computational agents, as well as the common agent model (CAM) from which the GDA model is derived.

4.3.1. Computational Agents

The use of artificial intelligence (AI) techniques for modelling places in VWs is a well-established practice in architecture. In particular, AI techniques can be used for designing accurate models of virtual places, testing the quality of these models and optimising or automating the design process (AEM, 2016). At the centre of this practice are computational agents (Poole, et al., 1998), which are entities that are capable of acting in some environment and whose decisions about their own actions can be explained in terms of computation (Poole & Mackworth, 2010). Fig. 4.5 shows a generic computational agent model.

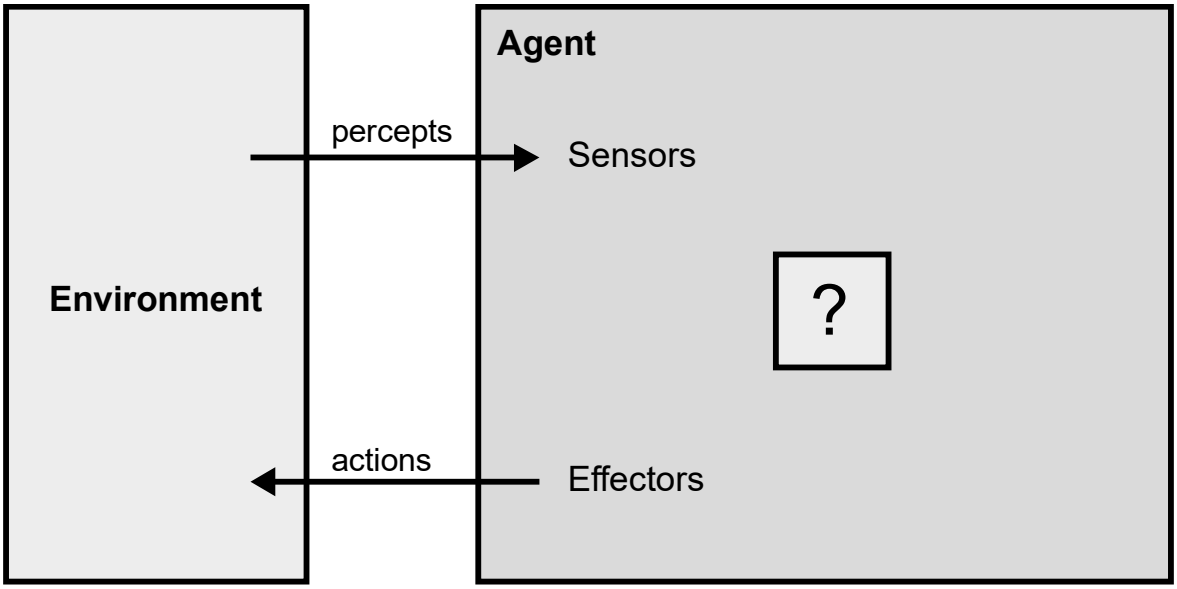


Figure 4.5. A generic computational agent model (Russell & Norvig, 2010)

In general, computational agents are expected to be able to perceive their environment (through sensors) and act on it (through effectors). However, depending on the nature of the environment (as well as other factors such as the complexity of the agent itself), they may also be able to act intelligently in many ways. For example, an agent may be able to reason about the condition of its environment and act based on condition-action rules. Fig. 4.6 shows a reflex agent model that is capable of exhibiting such behaviour.

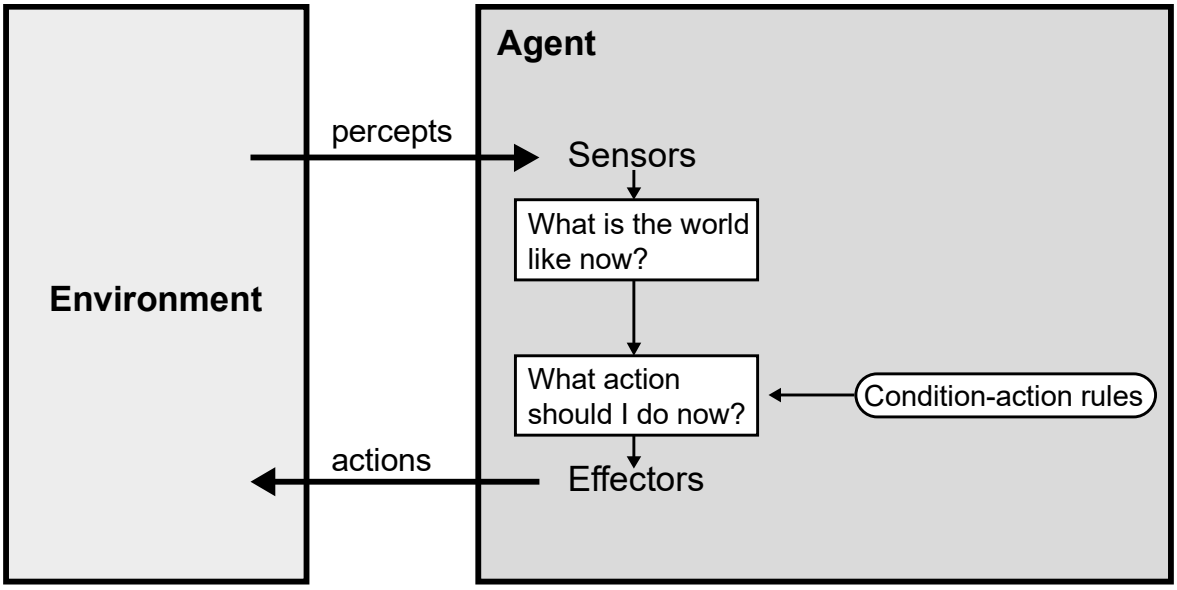


Figure 4.6. A reflex agent model (Russell & Norvig, 2010)

4.3.2. Common Agent Model

The GDA model derives from the CAM (Maher & Gero, 2002), which can be used to increase interactivity in VWs. When the CAM is applied to objects in VWs, it enables them to be able to automatically configure themselves as needed. Fig. 4.7 shows the CAM, whose architecture includes agent elements and computational processes for supporting agent-agent and agent-VW interactions.

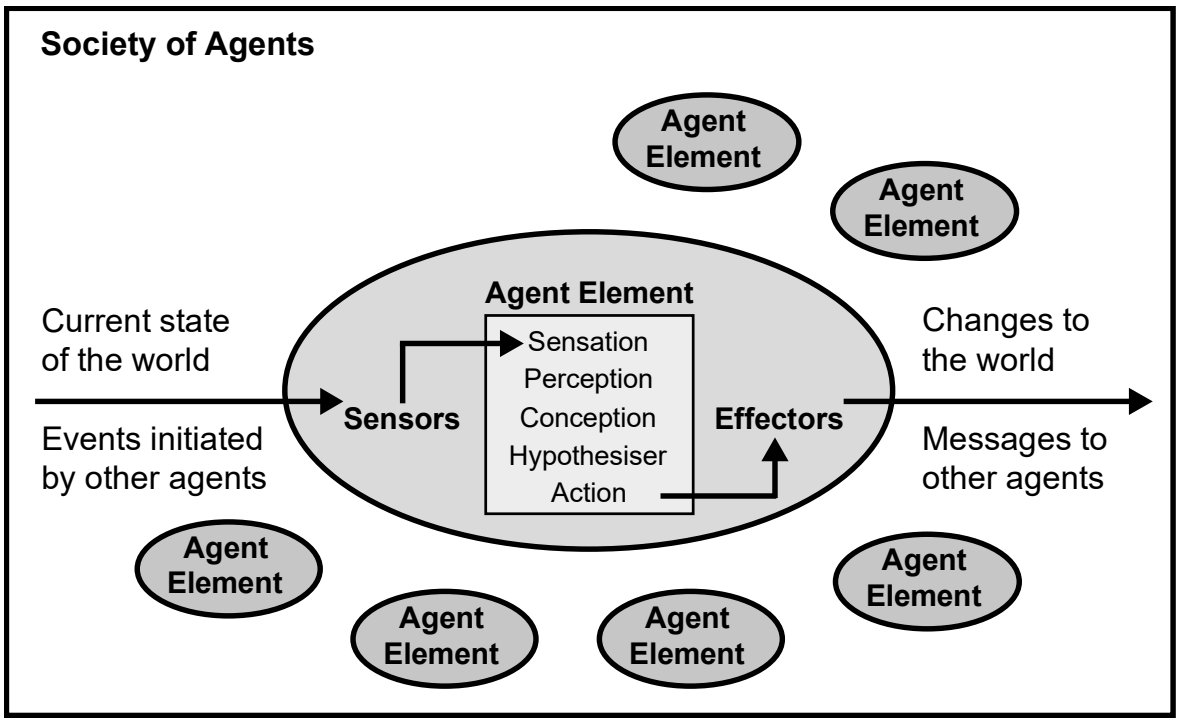


Figure 4.7. The CAM (Maher & Gero, 2002)

Using the CAM, objects in VWs such as doors, windows, lights, walls and rooms are represented as agent elements. Consequently, a VW application can be represented as a society of agents each of which is capable of sensing and acting in the VW by reasoning about its current state. The CAM provides a common vocabulary for describing, representing and implementing the knowledge and communication of an agent (Maher, et al., 2003). Therefore, its use to develop an agent enables the agent to interact with the VW (including other agents and objects) through sensors and effectors. Such an agent has a reasoning mechanism with five types of reasoning, which are as follows:

Sensation - transforming raw inputs from the sensors into data that is more appropriate for the agent to use for reasoning and learning

Perception - the process that finds grounded patterns of invariance in the agent's representation of the sense data for constructing concepts

Conception - learns about concepts and uses them to reinforce or modify the agent's beliefs and goals

Hypothesiser - identifies mismatches between the current state of the VW and its desired state and hypothesises goals in order to eliminate mismatches

Action - the process that reasons about the sequence of operations in the VW and enables the agent to achieve its goals.

4.4. Summary

This chapter presented the design and development of the VWADM using theory and design and development knowledge related to linguistic design in VWs, worldviews for designing in VWs and agent-based design in VWs. First, it discussed linguistic design in VWs. This was followed by a discussion of views for designing in VWs. Finally, a discussion was given about agent-based design in VWs.

5. VWADM: A New Method for Designing VW Applications

This chapter presents a description of the virtual world applications design method (VWADM), including its two components: the generative design grammar (GDG) framework and generative design agent (GDA) model. As such, the chapter begins with an overview of the composition of the VWADM. Next, it describes the GDG framework. The chapter finishes with a description of the GDA model.

5.1. Composition of the VWADM

The VWADM is a generative method for creating place models of VW applications during the design process. It combines the GDG framework and GDA model to establish the means for capturing spatial data about VW applications (including the objects that comprise them), transforming and storing such data as building information that is subsequently used to dynamically generate models of these applications. The GDG framework and GDA model are derived from methods developed by Maher et al and Gu and Maher to address the problem of static design in architecture (Maher, et al., 2000; Maher, et al., 2001; Maher & Gu, 2003; Gu & Maher, 2003; Gu & Maher, 2004). Together, the GDG framework and GDA model form the template of a mechanism that is used for engineering the design of VW applications prior to construction. Consequently, the VWADM is also a proactive method. Proactive methods are concerned with engineering the design of applications prior to construction (Moynihan, et al., 2001; Godwin, et al., 2010; Forcada, et al., 2015), which distinguishes them from other reactive methods (Freund, et al., 1998; Gianacakes, 2003) that engage with design late in the development process.

The GDG framework can be used to develop GDGs, which provide modelling concepts such as a set of grammars and rules for capturing and storing spatial data and transforming it into building information that is used to create conceptual models of VW applications. In complement, the GDA model can be used to develop GDAs, which wrap around GDGs and provide a way to automate the process of generating physical models of VW applications, whose specifications are based on the GDGs. Fig. 5.1 shows that the VWADM and the output of the GDG framework and GDA model (i.e., GDGs and GDAs) integrate neatly into the software development process.

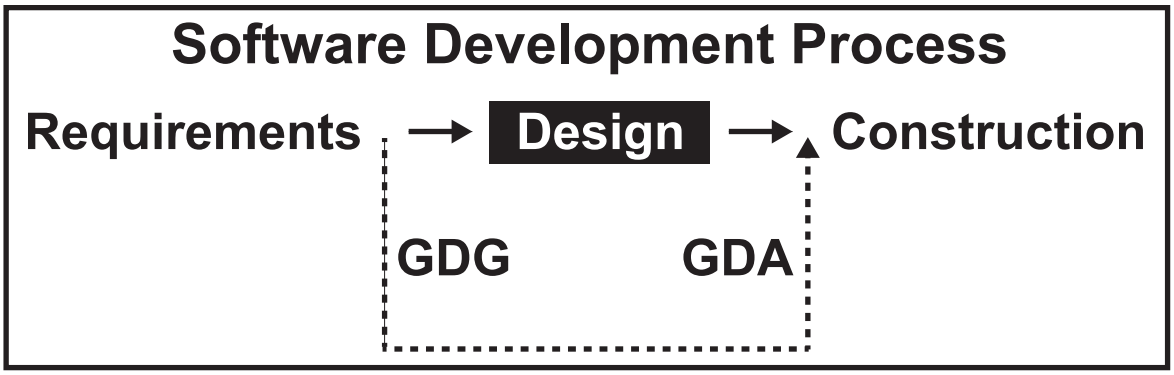
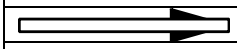


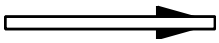
Figure 5.1. Integration of the VWADM into the software development process

The VWADM is comprised of a collection of symbols that are defined for use for describing GDGs and presenting them to stakeholders for review. Some of them are used for describing GDGs in the next section (and in the case studies that are presented in Chapter 7). Table 5.1 provides the meanings of these symbols, including those that are not used in the examples given in this thesis.

Table 5.1. A summary of the symbols used for describing and presenting GDGs

Symbol	Meaning
⇒ or →	Material implication or implies; if…then; from…to; maps to.
+	Registration mark used to show the position of shapes on the left-hand side and right-hand side of the rule, relative to each other.
•	Spatial label used to control the application of shape rules.
+	Additive symbol. It may be used in conjunction with state labels to indicate additive rules.
-	Subtractive symbol. It may be used in conjunction with state labels to indicate subtractive rules.

	Hyperlink symbol, which indicates that the means exists for teleportation or transfer from one location to the other.
	Hyperlink symbol, which indicates that the means exists for teleportation or transfer from one location to the other. Used with multilevel VW applications.
	Wayfinding aid; shows path in an area.
	Wayfinding aid; shows path between two adjacent areas.
♦	Opening that connects two adjacent areas.
a/b	Relative shape that indicates a multilevel area, which in this example is two levels (i.e., level a and level b).

The symbols that are not used in the design examples given in this thesis are the spatial label • and the wayfinding aids  and  . In the case of the spatial label, it is not used because the initial shape in each example serves as a marker during transformation[5]. In the case of the wayfinding aids, they were not used because each example uses teleportation as the mode of navigation (as opposed to pathfinding).

The GDG framework and GDA model are described in the following sections.

5.2. GDG Framework

The GDG framework provides guidelines and strategies for developing GDGs. It allows us to specify the general structure of a GDG and its basic components, which are the design rules. Fig. 5.2 shows the GDG framework.

[5] See (Hoisl & Shea, 2013) for a discussion on the use of markers in shape and spatial grammars.

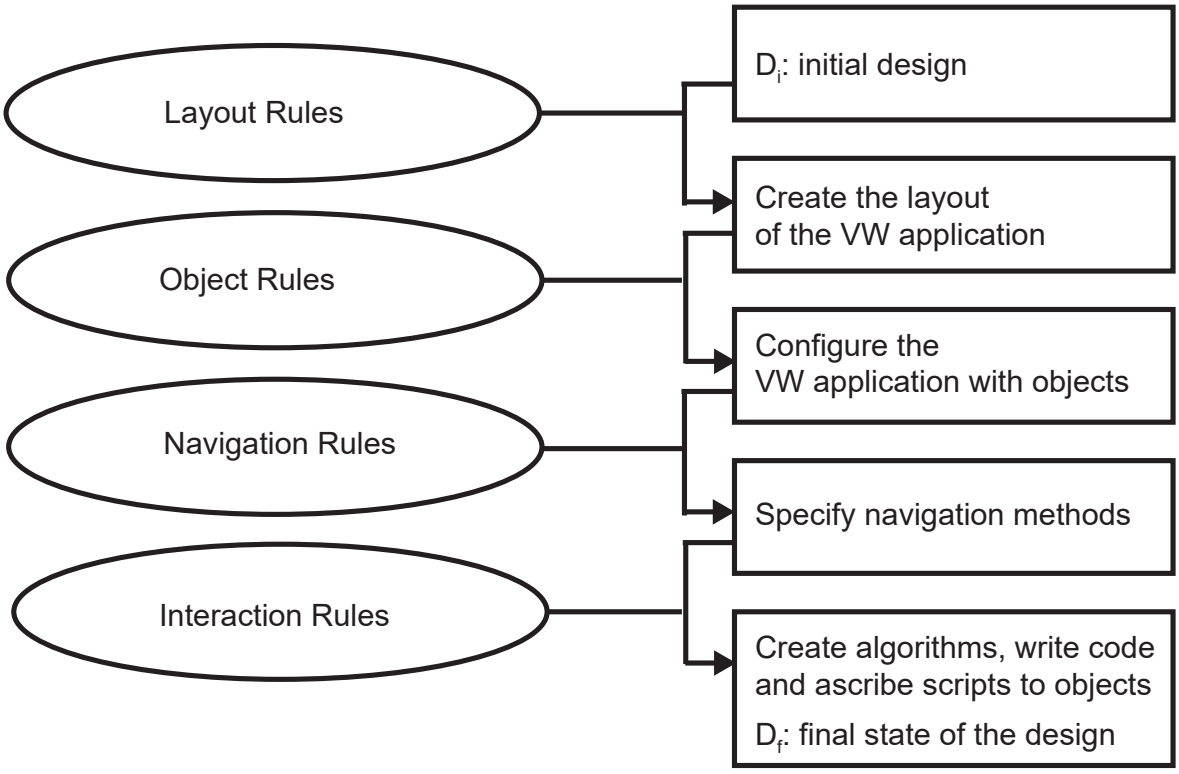


Figure 5.2. The GDG framework (Gu & Maher, 2005)

By using the GDG framework, GDGs can be developed for creating place-oriented conceptual models of VW applications that are intended to provide support for various in-world activities, and therefore reflect many different architectural styles.

A GDG is a set of rules that describe an architectural style. A GDG G is comprised of design rules R, an initial design D_i and a final state of the design D_f such that:

$$G = \{R, D_i, D_f\}.$$

The basic components of the GDG are the design rules R. The general structure of the GDG for creating the conceptual models of VW applications consists of four sets of design rules, which are (1) layout rules R_a, (2) object rules R_b, (3) navigation rules R_c, and (4) interaction rules R_d such that:

$$R = \{R_a, R_b, R_c, R_d\}.$$

The four sets of design rules correspond to the four phases for place-oriented design of VW applications, which are as follows:

Layout design: creating the layout of the VW application, where each area has a purpose that accommodates a certain set of intended activities.

Object design: configuring the VW application with objects that provide visual boundaries of the application, as well as visual cues for supporting the intended activities.

Navigation design: specifying navigation methods that use wayfinding aids such as hyperlinks and teleportation devices for assisting the movements of users (using avatars) between different areas of the VW application.

Interaction design: designing algorithms, writing code, and ascribing scripts to objects, to enable users to be able to interact with the VW application.

A GDGs design rule specifies that when the left-hand side object (LHO) is recognised in a VW and the state label (sL) matches, the right-hand side object (RHO) will replace the LHO such that:

$$LHO + sL \rightarrow RHO$$

State labels direct and ensure that the model of a VW application satisfies its design requirements.

In the following sections, we will use the example of a school application to further explain how the GDG's design rules can be used for creating the conceptual models of VW applications. For the purpose of creating the layout of the school application, the symbol  will be used to represent the main building (i.e., the initial design) and the symbol  will represent a set of stairs that lead into the main building. The symbol  will represent a classroom of which there are two that are located to the back of the main building and the symbol ┼ will represent the registration mark.

5.2.1. Layout Rules

The first set of rules to be applied using the GDG are layout rules, which create the layout of the VW application according to the type of activity it is intended to support.

Fig. 5.3 shows that after applying the GDG's layout rules, the RHO (i.e., the layout of the main building with a set of stairs attached to its front) will replace the LHO (i.e., the layout of the main building).

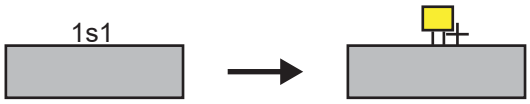


Figure 5.3. Layout rule for adding a set of stairs to the school's main building

The state label sL = 1 shows that we are working in the first phase of the GDG's design rules and sL= s1 is the design context, which indicates that the layout of a set of stairs, which lead into the main building, should be generated based on a certain specification s1.

Applying the layout rule for adding a set of stairs to the school's main building requires the following conditions to be met:

- The layout of the school's main building is recognised in the VW
- The design context (s1) matches some current design requirements or needs that are supposed to be used by designers (i.e., manually) or computational agents (i.e., automatically) to model the school application

Fig. 5.4 shows that after applying the GDG's layout rules, the RHO (i.e., the layout of the main building with a set of stairs attached to its front and a classroom added to the back of it) will replace the LHO (i.e., the layout of the main building with a set of stairs attached to it).

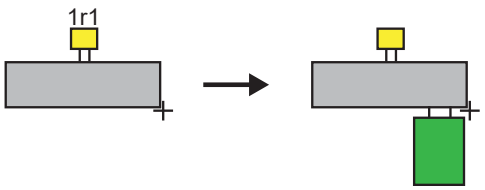


Figure 5.4. Layout rule for adding a classroom to the back of the main building

The state label sL = 1 shows that we are working in the first phase of the GDG's design rules and sL= r1 is the design context, which indicates that the layout of a room (i.e., a room whose purpose is to support learning activities) should be generated based on a certain specification r1.

Applying the layout rule for adding a classroom to the back of the main building requires the following conditions to be met:

- A layout that represents the school's main building registered with a set of stairs is recognised in the VW.
- The design context (r1) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school application.

5.2.2. Object Rules

The second set of rules to be applied using the GDG are object rules, which are used to configure the VW application with various objects that form visual boundaries as well as visual cues of the application for supporting the intended learning activities.

Fig. 5.5 shows that after applying the GDG's object rules for one of the classrooms of the school application, a visual boundary of it is generated.

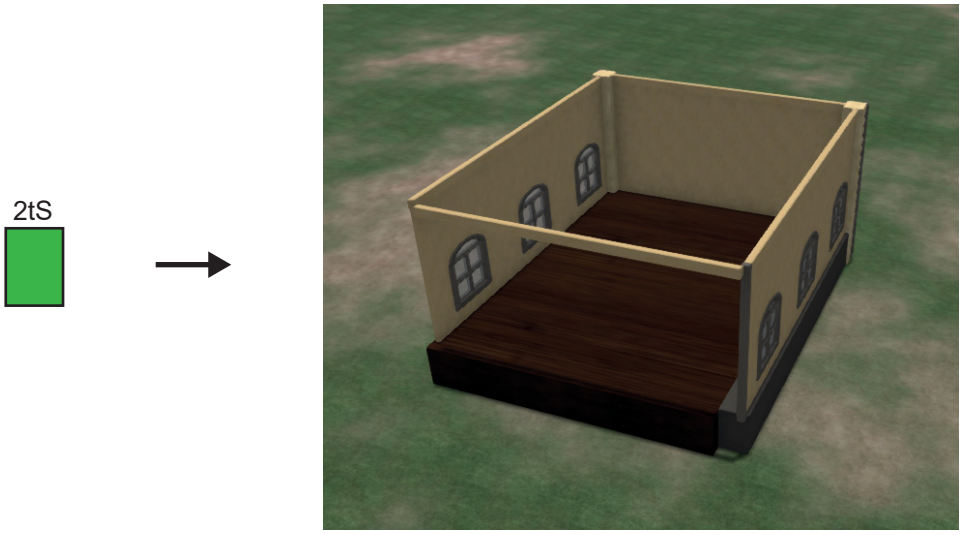


Figure 5.5. Object rule for configuring the visual boundary of a classroom

The state label sL = 2 shows that we are working in the second phase of the GDG's design rules and sL = tS is the design context, which indicates that a certain texture scheme should be applied to the walls of the classroom.

Applying the object rule for configuring the visual boundary of the classroom requires the following conditions to be met:

- The layout that was generated for the school application contains a main building with a set of stairs that is registered with a classroom to the back of it.
- The design context (tS) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school application.

Fig. 5.6 shows that after applying the GDG's object rules to one of the classrooms of the school application, it is also configured with visual cues (i.e., a blackboard, book case, books, chairs and desks) for supporting the intended learning activities.

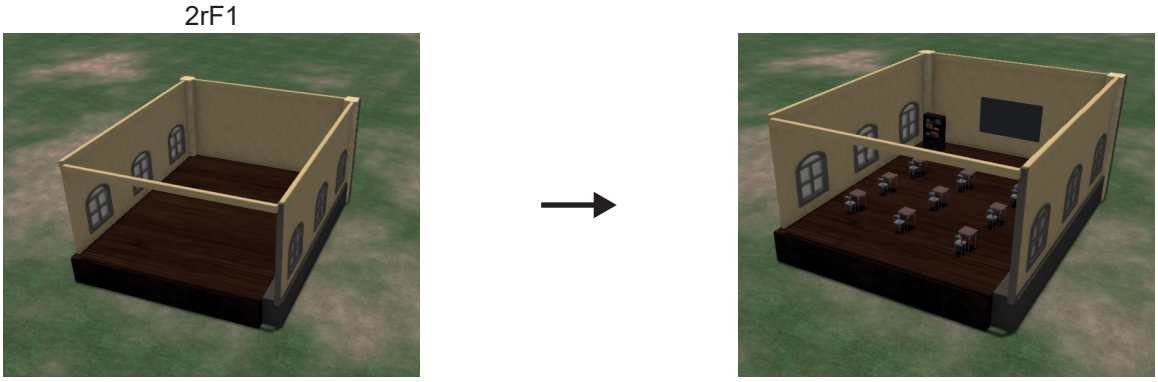


Figure 5.6. Object rule for configuring a classroom with visual cues

The state label sL = 2 shows that we are working in the second phase of the GDG's design rules and sL = rF1 is the design context, which indicates that a set of furniture should be generated in the classroom.

Applying the object rule for configuring the classroom with visual cues that support learning requires the following conditions to be met:

- A visual boundary of the classroom has been generated.

- The design context (rF1) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school application.

5.2.3. Navigation Rules

The third set of rules to be applied using the GDG are navigation rules, which are used to specify the VW application's navigation methods using wayfinding aids such as hyperlinks and teleportation devices.

Fig. 5.7 shows that after applying the GDG's navigation rule, the RHO (i.e., the model of the school with teleportation enabled between its two classrooms) will replace the LHO (i.e., the model of the school without teleportation enabled between its two classrooms).

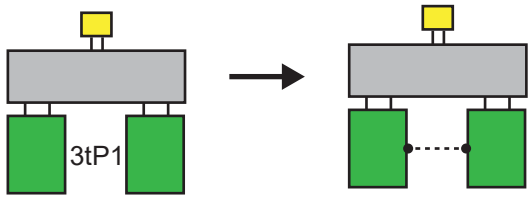


Figure 5.7. Navigation rule that specifies a teleport navigation method

The state label sL = 3 shows that we are working in the third phase of the GDG's design rules and sL = tP1 is the design context, which indicates that teleportation devices should be generated in order to enable movement from one classroom to the other.

Applying the navigation rule that specifies the school application's navigation method requires the following conditions to be met:

- Visual boundaries of the two classrooms have been generated.
- The two classrooms have been configured with all necessary visual cues.
- The design context (tP1) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school.

Fig. 5.8 shows the effect of applying the navigation rule, which generates a teleportation device in this example.

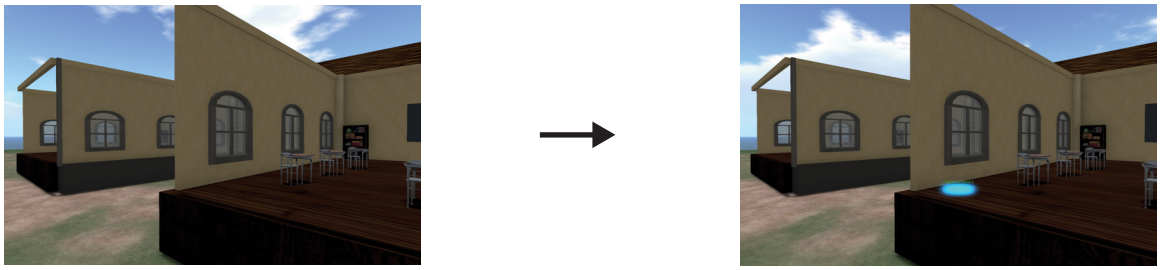


Figure 5.8. An example of the effect of applying a navigation rule that specifies a teleport navigation method

Both of the images above show the visual boundaries of the two classrooms of the school application. The image on the left is of the two classrooms prior to applying the GDG's navigation rules. The image on the right shows the two classrooms after applying the GDG's navigation rules and a teleportation device has been generated. By ascribing the appropriate scripted behaviour to the teleportation devices, users of the school application will be able to click on any one of them to teleport to the other classroom.

5.2.4. Interaction Rules

The fourth and final set of rules to be applied using the GDG are interaction rules, which are used for designing algorithms, writing code and ascribing scripts to objects to enable users to be able to interact with the VW application. As mentioned earlier, interaction rules are non-spatial and non-visual. Therefore, they are expressed using IF…THEN… statements. The following is an example of the IF…THEN… statements for the teleportation device in the school application:

sL = 4

IF: Visual boundaries of the two classrooms have been generated

AND

The two classrooms have been configured with all necessary visual cues

THEN: render teleport device

AND

Ascribe appropriate behaviour to teleport device according to design requirements

Fig. 5.9 shows an example of a VW application, which was constructed based on the conceptual model of the school application (it was generated by applying the GDG's four sets of design rules).

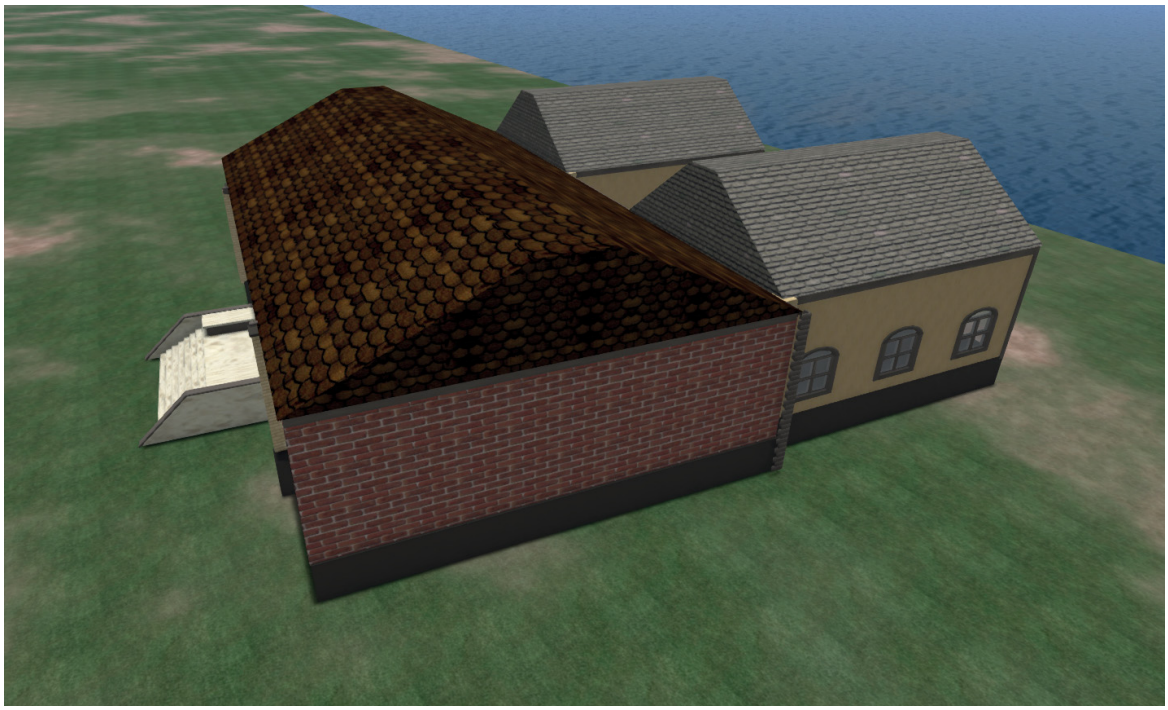


Figure 5.9. A VW application for supporting learning activities

The GDG framework provides guidelines and strategies for developing GDGs. It allows us to specify the general structure of a GDG and its basic components, which are the design rules. By using the GDG framework, GDGs can be developed for creating place-oriented conceptual models of VW applications with varying degrees of complexity, reflecting different architectural styles. For example, GDGs can be developed for creating models of VW applications such as a building with many differently shaped rooms that span multiple storeys. These models can then be reviewed by stakeholders. In order to generate physical models of VW applications, we can introduce the use of GDAs into the design process. A possible workflow would be to firstly develop a set of GDGs that can be used to generate a description of some VW application, which is also its design specification. Next, a GDA would read the design specification and use some form of reasoning to generate the physical model of the VW application.

5.3. GDA Model

The GDA model is a computational agent model that operates in a VW and has computational processes for reasoning, which can be used to automatically generate physical models of VW applications. The GDA's reasoning mechanism uses sensors and effectors as an interface between the GDA itself and the VW, as well as for modelling the VW application based on the models produced by the GDG (i.e., the design specification). Figure 5.10 shows the GDA model.

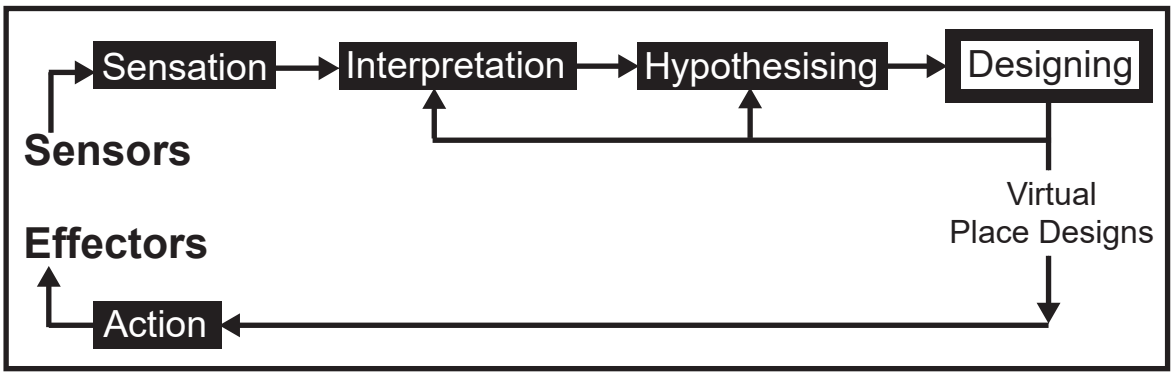


Figure 5.10. The GDA model

The GDA wraps around the GDG and senses and effects the properties of the models of VW applications. In addition, the GDA is capable of reasoning about the condition of its environment and act based on condition-action rules. The GDA's reasoning mechanism has five computational processes, which are as follows:

Sensation - using sensors to retrieve raw data from the VW to prepare for the process of interpretation.

Interpretation - interpreting the current design needs and the current state of the VW.

Hypothesising - setting up design goals that aim to eliminate mismatches between the current design needs in the VW and the current state of the VW.

Designing - reading the output of the GDG (i.e., the design specification) and using its description to provide the design of the VW application in order to satisfy current design goals.

Action - planning actions for implementing the design specification in the VW, as well as activating the planned actions in the VW.

The GDA's five computational processes form a recursive loop, in which new creations and changes in the VW trigger the GDAs to start a new cycle of reasoning and designing. This way, objects in the VW can be modelled and generated dynamically as needed.

5.3.1. Sensation

During the sensation process (see Fig. 5.11), the GDA retrieves raw data from the external world to prepare for the interpretation process, in which sense data is transformed in order to construct the GDA's interpreted world W_{int} such that:

$$W_{ext} \rightarrow W_{int}$$

$$W_{ext} = A_{ext} \cup E_{ext} \cup wA_{ext} \cup O_{ext}$$

$$W_{int} = A_{int} \cup E_{int} \cup wA_{int} \cup O_{int}$$

In addition, the GDA interprets the current design needs N in the VW and the current state sT of the VW such that:

$$N = \tau(W_{int})$$

$$sT \in W_{int}$$

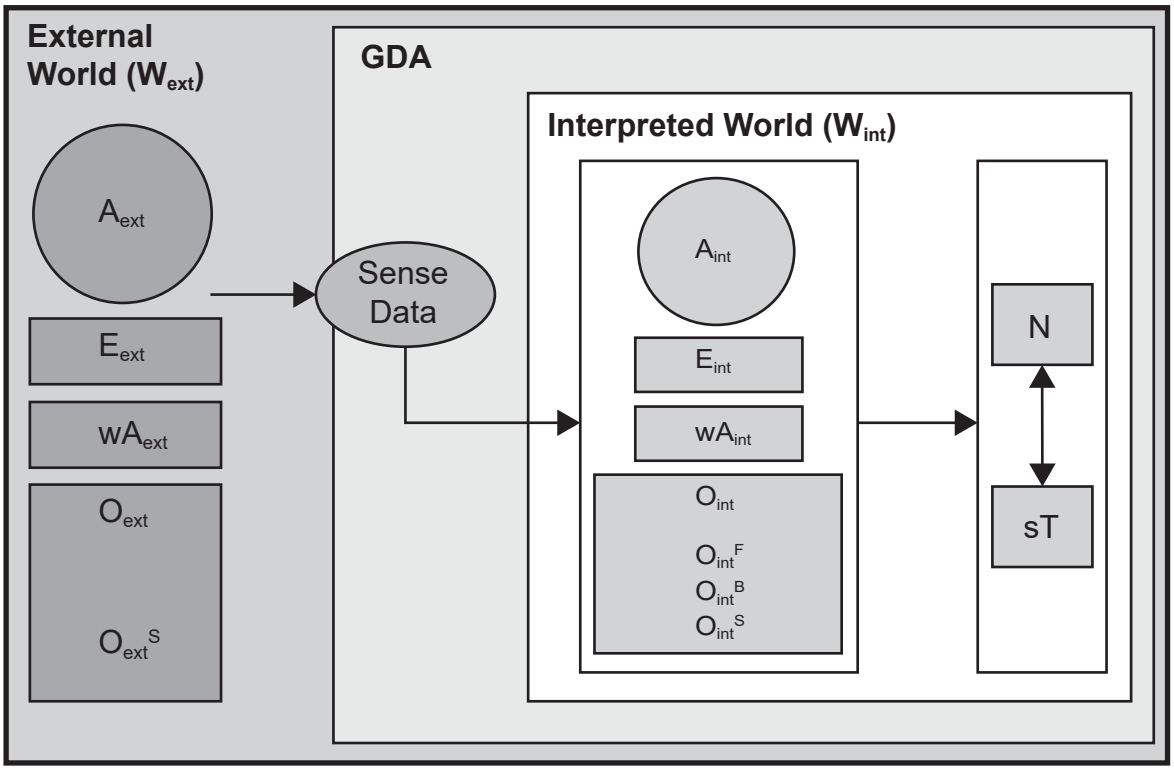


Figure 5.11. The GDA's sensation process

5.3.2. Interpretation

During the interpretation process (see Fig. 5.12), the interpreted world W_{int} is constructed based on the external world W_{ext} such that:

$$W_{ext} \rightarrow W_{int}$$

$$W_{ext} = A_{ext} \cup E_{ext} \cup wA_{ext} \cup O_{ext}$$

$$W_{int} = A_{int} \cup E_{int} \cup wA_{int} \cup O_{int}$$

The GDA's interpretation process has three sub-processes. In the first sub-process, the sense data from the external world W_{ext} is transformed into information that can be understood by the GDA such that:

$$A_{int} = \tau(A_{ext})$$

$$E_{int} = \tau(E_{ext})$$

$$wA_{int} = \tau(wA_{ext})$$

$$O_{int\_i}^{S} = \tau(O_{ext\_i}^{S})$$

In the second sub-process, for any object in the VW, the interpreted functions $O_{int\_i}^{F}$ and interpreted behaviours $O_{int\_i}^{B}$ are derived from the interpreted structures $O_{int\_i}^{S}$ such that:

$$O_{int\_i}{}^{B} = \tau(O_{int\_i}{}^{S})$$

$$O_{int\_i}{}^{F} = \tau(O_{int\_i}{}^{B})$$

In the third sub-process, the current design needs N in the VW and the current state sT of the VW are interpreted based on the information gained from the first two processes such that:

$$N = \tau(W_{int})$$

$$sT \in W_{int}$$

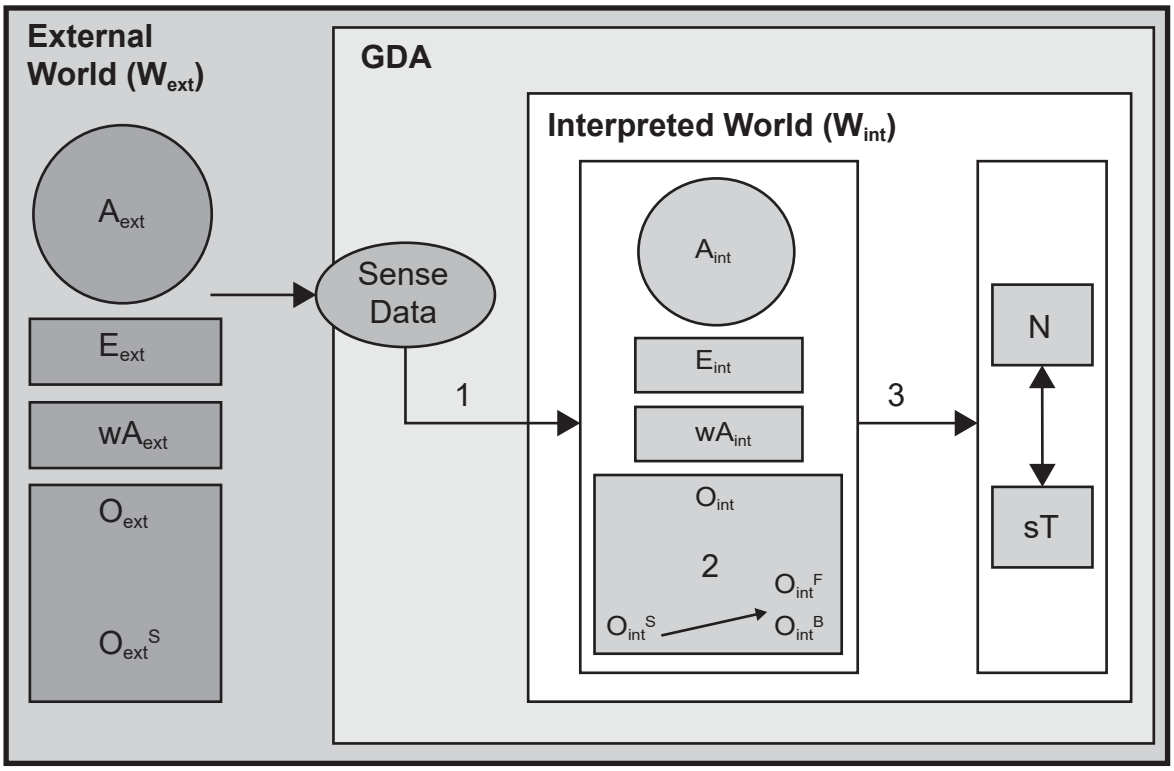


Figure 5.12. The GDA's interpretation process

5.3.3. Hypothesising

During the hypothesising process (see Fig. 5.13), the GDA sets up design goals in the expected world W_{exp} in order to eliminate mismatches between the current design needs N in the VW and the current state sT of the VW such that:

$$W_{int} \rightarrow W_{exp}$$

$$W_{exp} = A_{exp} \cup E_{exp} \cup O_{exp}$$

It is optional for an object in the GDA's expected world to have a counterpart in the interpreted world, since objects in VWs can be generated based either on existing structures or by new creations.

The GDA is capable of hypothesising three different types of goals. The first type are goals that are related to designing objects in VWs, which are represented by expected functions O_{exp}^{F} and expected behaviours O_{exp}^{B} such that:

$$O_{exp}^{F} = \{O_{exp\_1}^{F}, O_{exp\_2}^{F}, \ldots, O_{exp\_n}^{F}\}$$

$$O_{exp}^{B} = \{O_{exp\_1}^{B}, O_{exp\_2}^{B}, \ldots, O_{exp\_n}^{B}\}$$

for any expected function $O_{exp\_i}^{F}$ and expected behaviour $O_{exp\_i}^{B}$ such that:

$$O_{exp\_i}^{F} \in \tau(N, sT)$$

$$O_{exp\_i}^{B} = \tau(O_{exp\_i}^{F})$$

The second type of goals that the GDA is capable of hypothesising are those that are used for initiating changes of properties of avatars in the VW. The third type of goals that the GDG is capable of hypothesising are those that are used for initiating events in the VW such that:

$$A_{exp} \in \tau A(N, sT)$$

$$E_{exp} \in \tau E(N, sT)$$

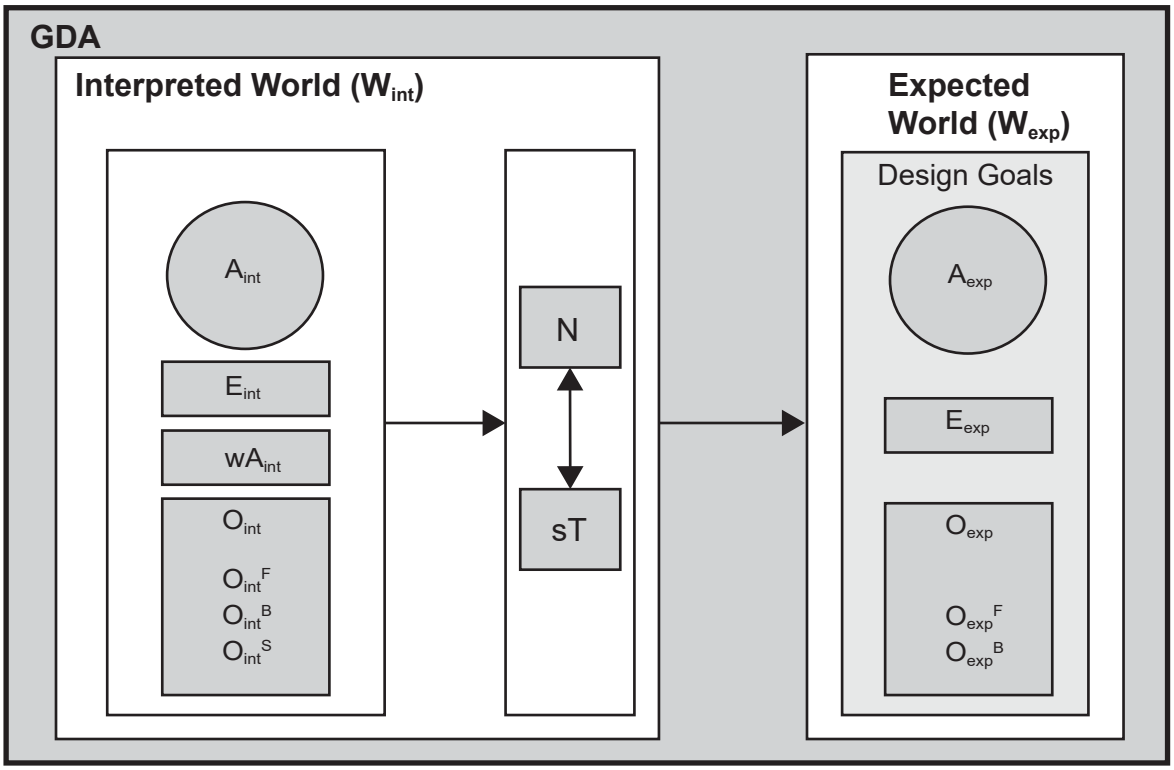


Figure 5.13. The GDA's hypothesising process

5.3.4. Designing

During the designing process, the GDA generates an object in order to satisfy a set of requirements or design goals. The object is represented by O_{exp}^{S} such that:

$$O_{exp}^{S} = \tau(O_{exp}^{F}, O_{exp}^{B})$$

The GDA's generative capability enables objects to be generated dynamically as needed. The GDA's design component is supported by the application of a GDG in order to generate the objects in the VW.

5.3.5. Action

During the action process (see Fig. 5.14), the GDA carries out action planning and action activation. In action planning, the GDA plans actions for generating the model of an object O_{exp}^{S} based on the GDG's specification and for realising other initiated changes A_{exp} and E_{exp} in the VW. In action activation, the planned actions are activated in the VW by the GDA through the use of its effectors.

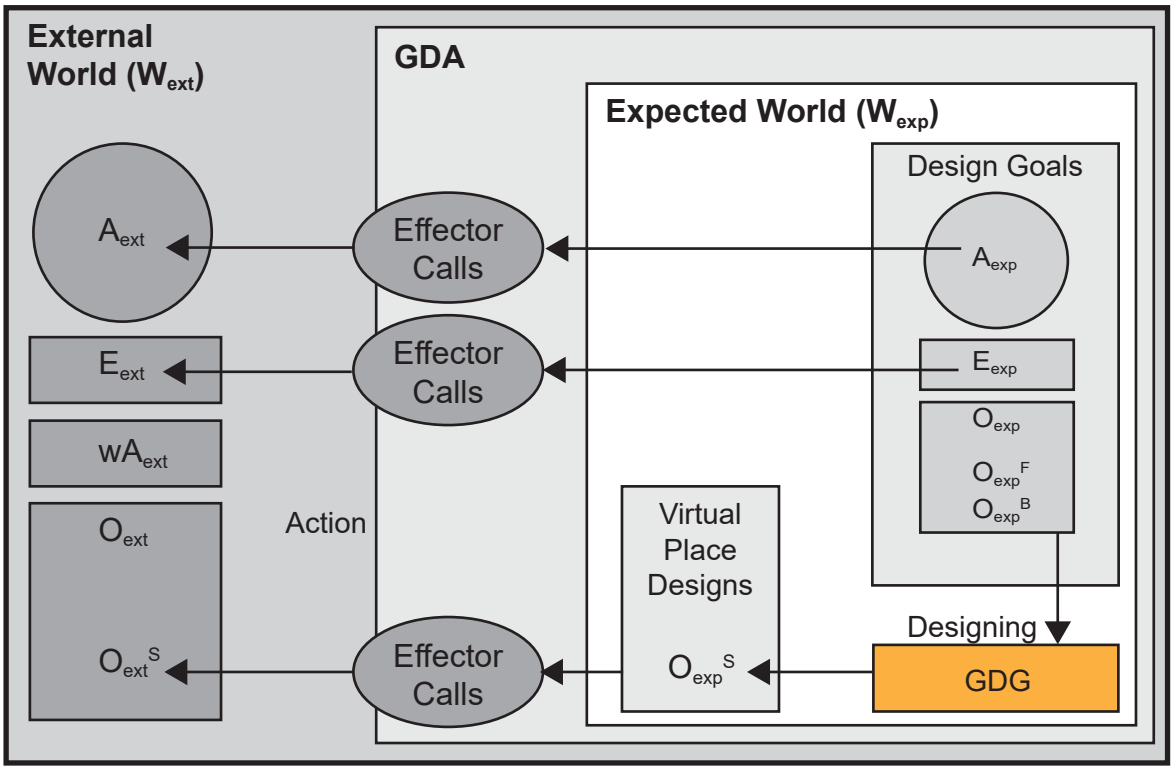


Figure 5.14. The GDA's design and action processes

The GDA model is a computational agent model that operates in a VW and has computational processes for reasoning, which can be used to automatically generate place-based models of VW applications. GDAs are wrappers for GDGs that use sensors and effectors to dynamically generate objects in VWs. These objects may be a set of assets for modelling VW applications or they may be the VW application itself (including the objects that comprise it). A GDA is capable of reasoning about the condition of its environment and act based on condition-action rules, which it applies recursively until it meets its design goals.

5.4. Summary

This chapter presented the VWADM and its two components, the GDG framework and GDA model. Firstly, the chapter gave an overview of the composition of the VWADM. This was followed by a description of the GDG framework. Finally, the chapter also provided a description of the GDA model.

6. Implementation of the VWADM

This chapter provides an overview of software called the Active Worlds (AW) agent package, which was developed to serve as a proof of concept for the VWADM. The chapter firstly discusses the technologies used for developing the AW agent package. It then continues on to describe the overall design of the AW agent package. Next, the chapter provides a description of the main components of the AW agent package. This is followed by a description of the architecture of the AW agent package and an explanation of its basic functionality. The chapter finishes with a brief discussion of two user guides that are provided for using the AW agent package to implement and extend the functionality of agents.

6.1. Technologies used for the AW Agent Package

The AW agent package is an implementation of the VWADM's mechanism for designing VW applications. It uses Jess and Java sensors and effectors for coding agent reasoning as rules. In addition, it uses an SDK called the AW SDK, which agents rely on for communicating with Active Worlds servers.

6.1.1. Jess

Jess is a small, light and fast rule engine and scripting environment for developing Java software that has the capacity to reason using knowledge supplied in the form of declarative rules (Friedman-Hill, 2008). It is generally used for the automation of expert systems as well as to develop intelligent agent systems.

The Jess rule engine employs the Rete algorithm (a graph structure of rules that supports rapid processing via non-iterative evaluation)[6] and it can be used to efficiently create Java applets and servlets and Enterprise JavaBeans.

Jess' scripting environment allows the use of either the Jess language, which is a highly specialised form of Lisp, or the Java programming language for coding rules. Jess enables the integration of both the Jess language and Java in such a way that Java functions can be accessed via the Jess language and vice versa.

Jess is written in Java and requires a Java Virtual Machine (JVM) to function. As such, both the Jess language and Java are native programming languages for it.

6.1.2. Java

Java is a general-purpose computer programming language that is concurrent, class-based and object-oriented (Gosling, et al., 2015). It is used to develop portable software that can run securely on any platform that supports Java without the need for recompilation.

As a result of its portability and security features, Java is well-suited for developing computer programs that operate in networked environments (Gosling & McGilton, 1996; Higuera-Toledano, 2017). Since the agents that were implemented in this research communicate over a network with an Active Worlds server, Java provides the perfect companion for Jess, whose state could be packaged up at an origin, sent across a wire and reconstituted at the destination.

[6] See (Forgy, 1989) for a further discussion of the Rete algorithm and its use for solving the many pattern/many object pattern match problem.

6.1.3. AW SDK

The AW SDK is a set of software development tools that provides an API for developing client applications, which can communicate with Active Worlds servers and function within the VWs generated by these servers (ActiveWorlds Inc., 2014). The core component of the AW SDK is the aw.dll file, which is a Microsoft Windows dynamic-link library (DLL) file that implements the Active Worlds client-server protocol.

In order to develop an application using the AW SDK, a programmer simply writes a C program which includes the header file aw.h and links to the import library aw.lib. The compiled executable can be run on any computer from anywhere as long as that computer has a network connection to the Active Worlds servers and aw.dll is available on it.

The AW SDK can be used to develop applications such as an automated program that explores and creates objects in VWs generated by Active Worlds servers. It can also be used to develop bots or non-playable characters (NPCs), which are typically avatars that are controlled by computer programs rather than humans (Schatten, et al., 2017). Administrators may also use the AW SDK to develop utilities for managing their VWs in Active Worlds.

6.2. Overall Design of the AW Agent Package

The AW agent package was designed to provide a flexible framework in which agents are the basis for all of the elements of a VW in Active Worlds. By using intelligent agent models, it becomes possible to dynamically generate such elements. The AW agent package is based on a set of abstract classes that form the generic architecture for constructing a multi-agent system (MAS) of agents. A MAS is an aggregation of agents that share a common connection with a VW (Wooldridge, 2009). Fig. 6.1 shows an entity relationship diagram (ERD) that depicts the concept of a MAS and agents.

Figure 6.1. MAS and agents

6.3. Main Components of the AW Agent Package

Agents usually share some ontological connection such as a room agent plus a set of wall agents that collectively comprise a virtual conference room (Maher, et al., 2003; Maher, et al., 2003). In complement, the MAS manages computational resources such as the connection to the VW, on behalf of the agents. Each entity that is capable of reflexive, reactive or reflective behaviours is modelled as an agent. Such an agent inherits these three generic behaviours (including an optional 3D representation) and can dynamically change the 3D representation and generate non-visual behaviours. The procedure to assert an object into a VW in Active Worlds is to copy an existing object, move it to a desired location and edit a dialog box to specify its properties. The procedure to create an agent in the VW is to configure it as a set of sensors and a set of rules. An agent can be configured using an XML file that is loaded through the use of a validating parser, rather than being statically linked at compile time. This flexible method enables the reconfiguration of agents running in the VW without having to recompile and restart the server. Fig. 6.2 shows the components that comprise an agent (or a ReteAgent).

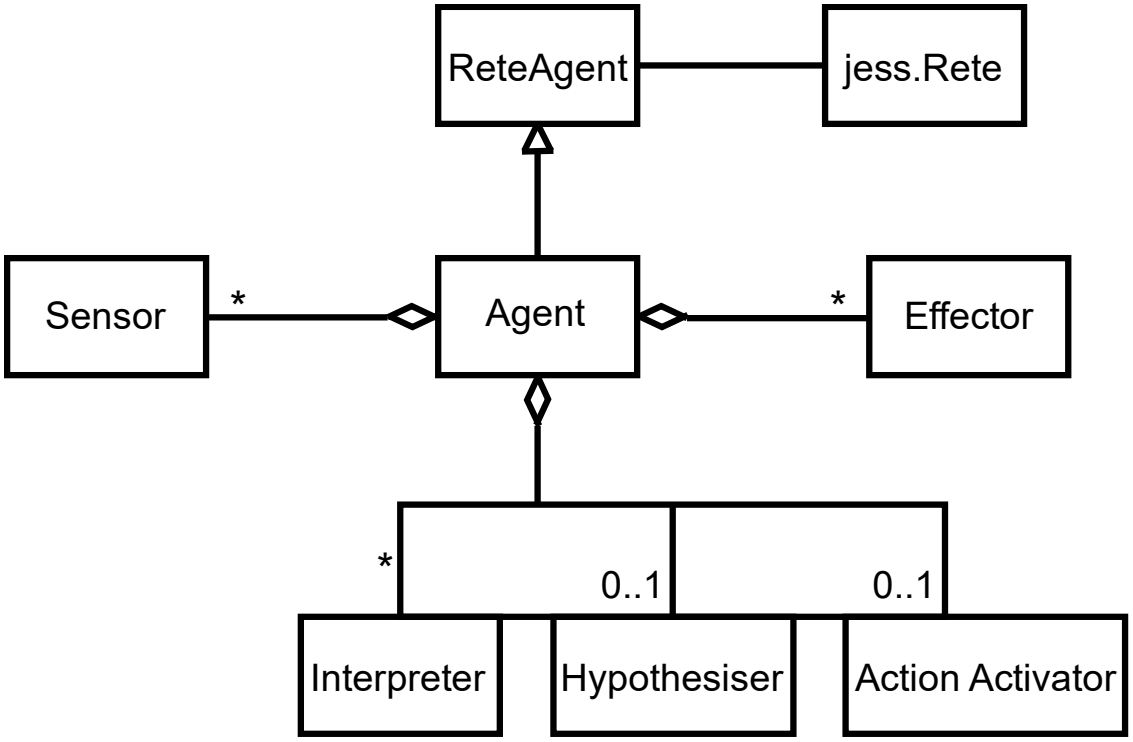


Figure 6.2. Components of an agent/ReteAgent

6.4. Architecture of the AW Agent Package

The VWADM's AW agent package implements sensors as java objects that sense an agent's environment. Effectors act on the environment. Both sensors and effectors use a Java Native Interface (JNI) to the AW SDK. An effector is a function call from the right-hand side of a rule. ReteAgent is an implementation of an agent that uses Jess for everything except sensors and effectors. Fig. 6.3 shows the architecture of ReteAgent.

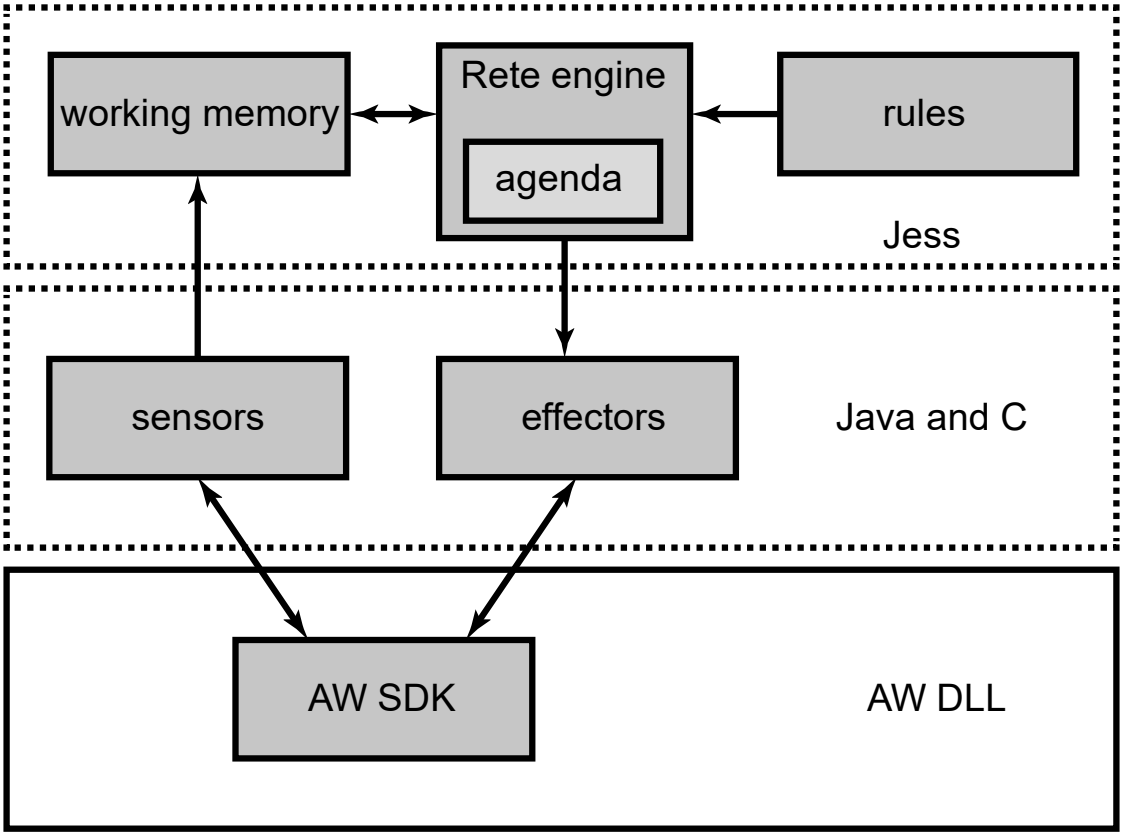


Figure 6.3. Architecture of an agent/ReteAgent

6.5. Basic Functionality of the AW Agent Package

A MAS can contain one or more agents that are instances of ReteAgent. The creator of a ReteAgent configures the agent by specifying a set of sensors, whose data is stored in Jess' working memory. Sensors also record messages from other agents that is stored in Jess' working memory. Each time new sense data is received, a new fact in the form of a Java bean is asserted into Jess' working memory. Fig. 6.4 shows an example of how a MAS communicates with the Active Worlds server.

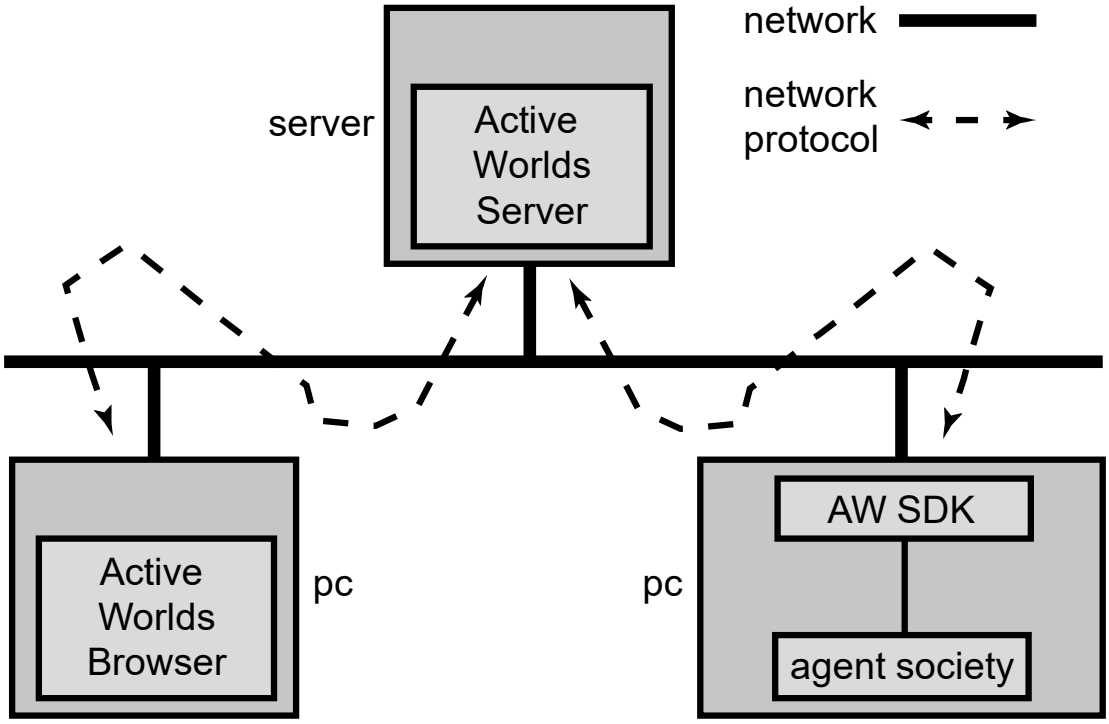


Figure 6.4. MAS communication

6.6. Guides for using the AW Agent Package to Implement and Extend the Functionality of Agents

The AW agent package can be used to implement agents using Java sensors and effectors and the Jess language for coding their reasoning as rules in Jess. A basic user guide is provided that discusses the implementation of the sensors and effectors for such agents (see Appendix C). The sensors and effectors are class-based. Therefore, developers can write their own sensors and effectors by extending the classes in the vwadm.awa.base Java package. An advanced user guide is provided that discusses how to do so (see Appendix D). The advanced user guide also discusses how to extend the functionality of agents to enable them to communicate with each other as well as for them to be able to dynamically create other agents. Such functionality is important to a MAS because it enables agents to enlist the support of other agents to achieve goals, commit to the execution of actions and report on progress.

6.7. Summary

This chapter provided an overview of the AW agent package. It firstly discussed the technologies used for developing the AW agent package. It then continued on to describe the overall design of the AW agent package. Next, the chapter provided a description of the main components of the AW agent package. This was followed by a description of the architecture of the AW agent package and an explanation of its basic functionality. Finally, the chapter briefly discussed two user guides that have been provided for using the AW agent package to implement and extend the functionality of agents.

7. Demonstration of the Functionality of the VWADM

This chapter presents three case studies that are used to demonstrate the functionality of the VWADM. Together, the case studies show how a GDA (which wraps around and encapsulates GDGs) can be used to dynamically generate objects in a VW by interpreting and applying the design rules that pertain to GDGs. The chapter begins with a presentation of the first case study, which demonstrates the use of the VWADM to design an office application. Next, it presents the second case study, which demonstrates the use of the VWADM to design a living space application. This is followed by a presentation of the third case study, which demonstrates the use of the VWADM to design a student centre application. All three of the case studies are intended to show that the VWADM can be used to create place models of VW applications that are varied in complexity. The chapter finishes with a review of some extended features and capabilities of the VWADM in light of the case studies.

7.1. Designing an Office Application

For the purpose of designing the office application, the following symbols and their meanings (Table 7.1) will be used:

Table 7.1. A summary of the symbols used in the office application and their meanings

Symbol	Meaning
	An office
	A meeting room to be added adjacent to the office

7.1.1. Layout Rules

The first set of rules that the GDA applies are the GDG's layout rules, which are used to create the layout of the office application.

Fig. 7.1 shows that after the GDA applies the GDG's layout rules, the RHO (i.e., the layout of the office with the meeting room added adjacent to it) will replace the LHO (i.e., the layout of the office)

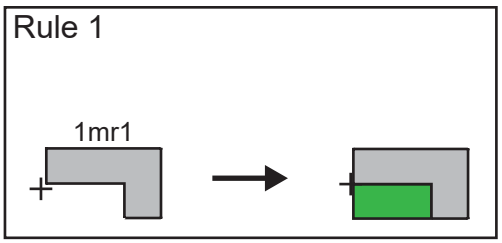


Figure 7.1. Layout rule for adding a meeting room adjacent to the office

The state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= mr1 is the design context, which indicates that the layout of a meeting room should be generated based on a certain specification mr1. The GDA matches state label sL = mr1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = mr1 to the current design goals, the GDA firstly interprets that some persons A_{int} are indicating that they would require a meeting room and a layout of the meeting room does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = mr1 (i.e., the layout of a meeting room is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for adding the layout of a meeting room next to the layout of the office requires the following conditions to be met:

- The layout of the office is recognised in the VW.

- The design context (mr1) matches some current design requirements or needs that are supposed to be used by the GDA to model the office application.

7.1.2. Object Rules

The second set of rules that the GDA applies are the GDG's object rules, which are used to configure the office application with various objects that form its visual boundaries. Such objects also provide visual cues for the types of activities that the office supports, people's notion of the place as an office and their orientation around it.

Fig. 7.2 shows that after the GDA applies the GDG's object rules for the office, a visual boundary of it is generated. The same figure (Fig. 7.2) also shows that after the GDA applies the GDG's object rules for the meeting room, a visual boundary of it is generated adjacent to the office.

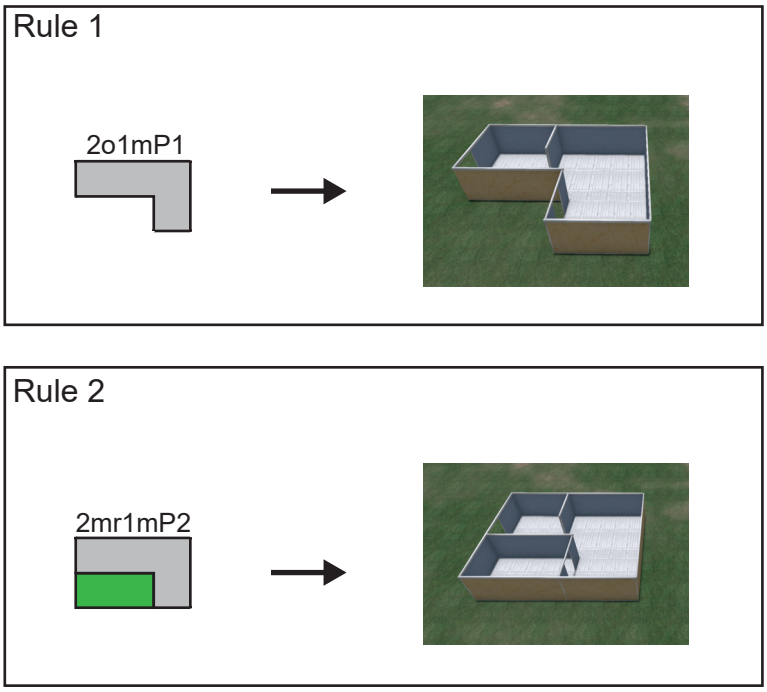


Figure 7.2. Object rules for configuring the visual boundary of the office application

In Rule 1, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= o1mP1 is the design context, which indicates that a certain matte paint should be applied to the interior walls of the office. The GDA matches

state label sL = o1mP1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = o1mP1 to the current design goals, the GDA firstly interprets that some persons A_{int} are indicating that they would require the interior walls of the office to have a certain matte paint colour and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = o1mP1 (i.e., a certain matte paint colour scheme for the interior walls of the office is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 1 for generating a matte paint scheme for the interior walls of the office requires the following conditions to be met:

- The visual boundary of the office is recognised in the VW.
- The design context (o1mP1) matches some current design requirements or needs that are supposed to be used by the GDA to model the office application.

In Rule 2, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= mr1mP2 is the design context, which indicates that a certain matte paint should be applied to the interior walls of the meeting room. The GDA matches state label sL = mr1mP2 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = mr1mP2 to the current design goals, the GDA firstly interprets that some persons A_{int} are indicating that they would require the interior walls of the meeting room to have a certain matte paint colour and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = o1mP1 (i.e., a certain matte paint colour scheme for the interior walls of the meeting room is needed).

Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 2 for generating a matte paint scheme for the interior walls of the meeting room requires the following conditions to be met:

- The visual boundary of the meeting room is recognised in the VW.
- The design context (mr1mP2) matches some current design requirements or needs that are supposed to be used by the GDA to model the office application.

Fig. 7.3 shows that after the GDA applies the GDG's object rules to the office, it is also configured with visual cues (i.e., chairs, tables, water coolers, etc.) for the types of activities that it supports, people's notion of the place as an office and their orientation around it. The same figure (Fig. 7.3) also shows that after the GDA applies the GDG's object rules to the meeting room, it is configured with visual cues (i.e., chairs, a large table in the form of an arc and a water cooler) for the types of activities that it supports, people's notion of the place as a meeting room and their orientation around it.

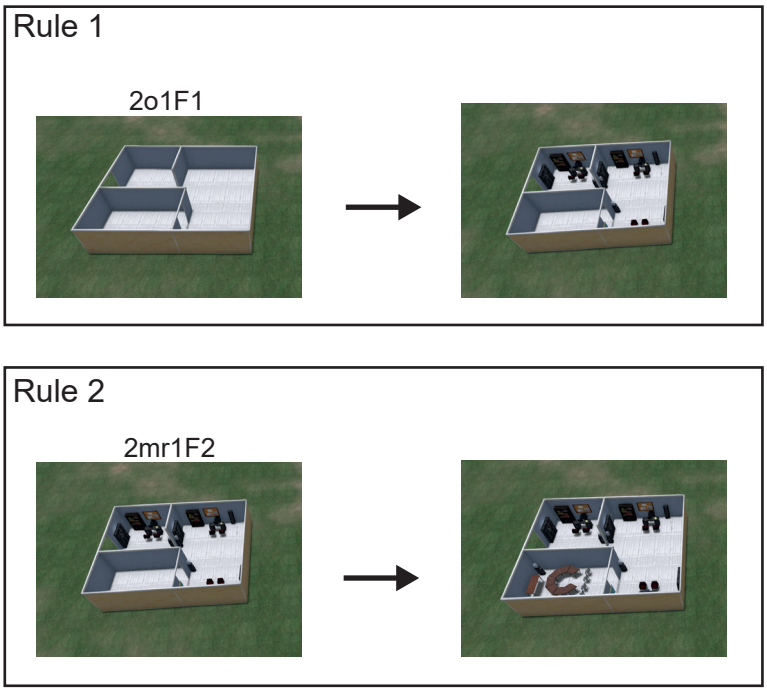


Figure 7.3. Object rules for configuring the office application with visual cues

In Rule 1, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= o1F1 is the design context, which indicates that a set of furniture should be generated in the office. The GDA matches state label sL = o1F1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = o1F1 to the current design goals, the GDA firstly interprets that some persons A_{int} are indicating that they would require certain furniture in the office and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = o1F1 (i.e., certain furniture in the office is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 1 for generating a set of furniture in the office requires the following conditions to be met:

- The visual boundary of the office is recognised in the VW.
- The design context (o1F1) matches some current design requirements or needs that are supposed to be used by the GDA to model the office application.

In Rule 2, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= mr1F2 is the design context, which indicates that a set of furniture should be generated in the meeting room. The GDA matches state label sL = mr1F2 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = mr1F2 to the current design goals, the GDA firstly interprets that some persons A_{int} are indicating that they would require certain furniture in the meeting room and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = mr1F2 (i.e., certain furniture in the meeting room is

needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 2 for generating a set of furniture in the meeting room requires the following conditions to be met:

- The visual boundary of the meeting room is recognised in the VW.
- The design context (mr1F2) matches some current design requirements or needs that are supposed to be used by the GDA to model the office application.

7.1.3. Navigation Rules

The third set of rules that the GDA applies are the navigation rules, which are used to configure the office application with wayfinding aids such as hyperlinks and teleportation devices.

Given the close proximity of the spaces of the office, it is not necessary to implement wayfinding aids. However, Fig. 7.4 shows a possible navigation rule for teleportation between the office and the meeting room.

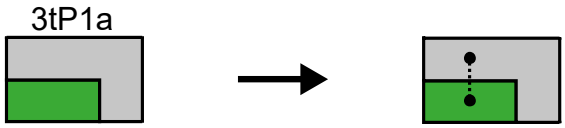


Figure 7.4. Navigation rule that establishes teleportation in the office application

After applying this navigation rule, the RHO (i.e., the model of the office with teleportation enabled between it and the meeting room) will replace the LHO (i.e., the model of the office without teleportation enabled between it and the meeting room).

The state label sL = 3 shows that we are working in the third phase of the GDG's design rules and sL = tP1a is the design context, which indicates that a teleportation device should be generated in order to allow movement between the office and the meeting room.

Applying the navigation rule that configures the office with wayfinding aids requires the following conditions to be met:

- Visual boundaries of the office and the meeting room have been generated.
- The office and the meeting room have been configured with all necessary visual cues.
- The design context (tP1a) matches some current design requirements or needs that are supposed to be used by the GDA to model the office application.

7.1.4. Interaction Rules

The fourth set of rules that the GDA applies are interaction rules, which are used for designing algorithms, writing code and ascribing scripts to objects to enable users to be able to interact with the office application. The following are the IF…THEN… statements that could be used to implement teleportation in the office application.

sL = 4

IF: Visual boundaries of the office and the meeting room have been generated

AND

The office and the meeting room have been configured with all necessary visual cues

THEN: render teleport devices in both the office and the meeting room

AND

Ascribe appropriate behaviour to teleport devices according to design requirements

Fig. 7.5 shows the final design of the office application, which was creating using the rules (i.e., layout design rules, object design rules, navigation design rules and interaction design rules) that have been described in the preceding sections (i.e., Sections 7.1.1 to 7.1.4).

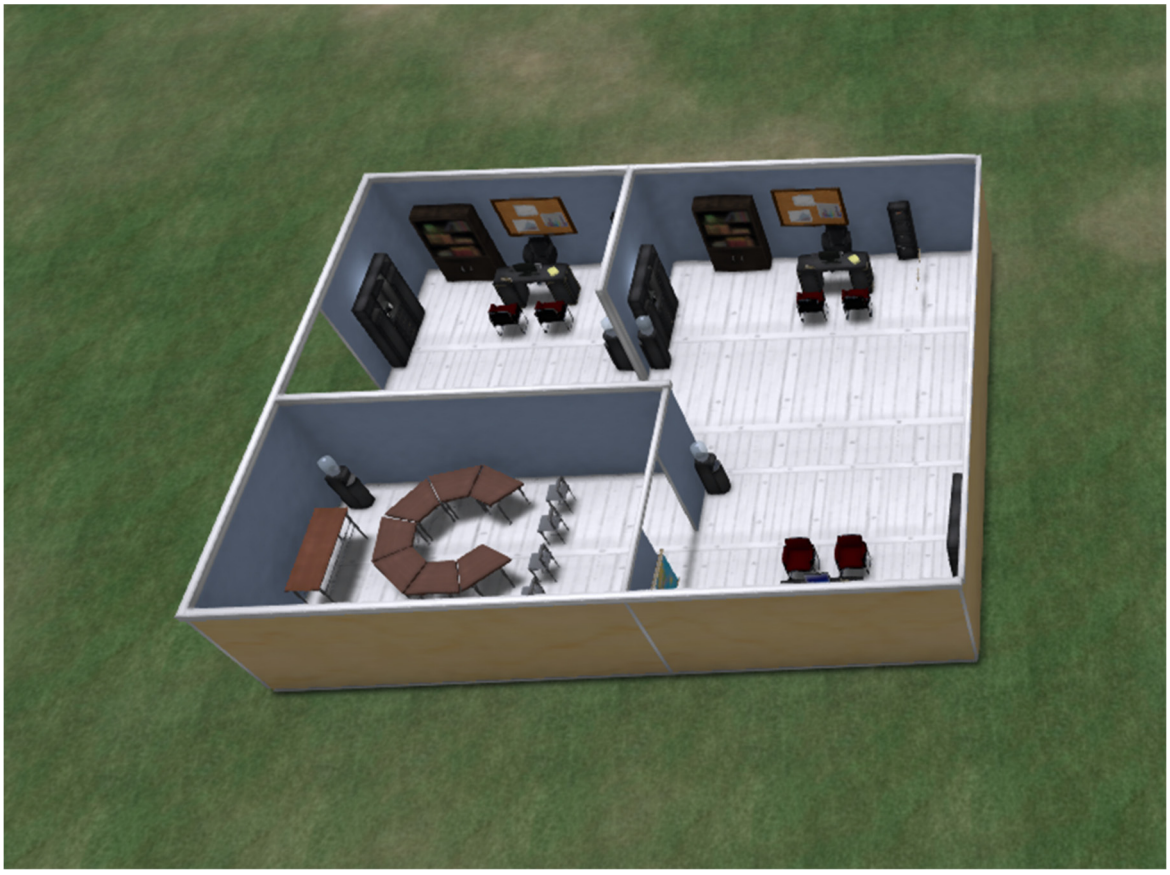


Figure 7.5. The final design of the office application

7.2. Designing a Living Space Application

For the purpose of designing the living space application, the following symbols and their meanings (Table 7.2) will be used:

Table 7.2. A summary of the symbols used in the living space application and their meanings

Symbol	Meaning
	A living room
	A bedroom
	A kitchen
	A bathroom

7.2.1. Layout Rules

The first set of rules that the GDA applies are the GDG's layout rules, which are used to create the layout of the living space.

Fig. 7.5 shows that after the GDA applies the GDG's first set of layout rules, the RHO (i.e., the layout of the living room with the addition of a bedroom) will replace the LHO (i.e., the layout of the living room). The same figure (Fig. 7.5) shows that after the GDA applies the GDG's second set of layout rules, the RHO (i.e., the layout of the living room with the bedroom and the addition of a kitchen) will replace the LHO (i.e., the layout of the living room with the addition of a bedroom). Finally, the figure (Fig. 7.5) shows that after the GDA applies the GDG's third set of layout rules, the RHO (i.e., the layout of the living room with the bedroom and kitchen and the addition of a bathroom) will replace the LHO (i.e., the layout of the living room with the bedroom and addition of a kitchen).

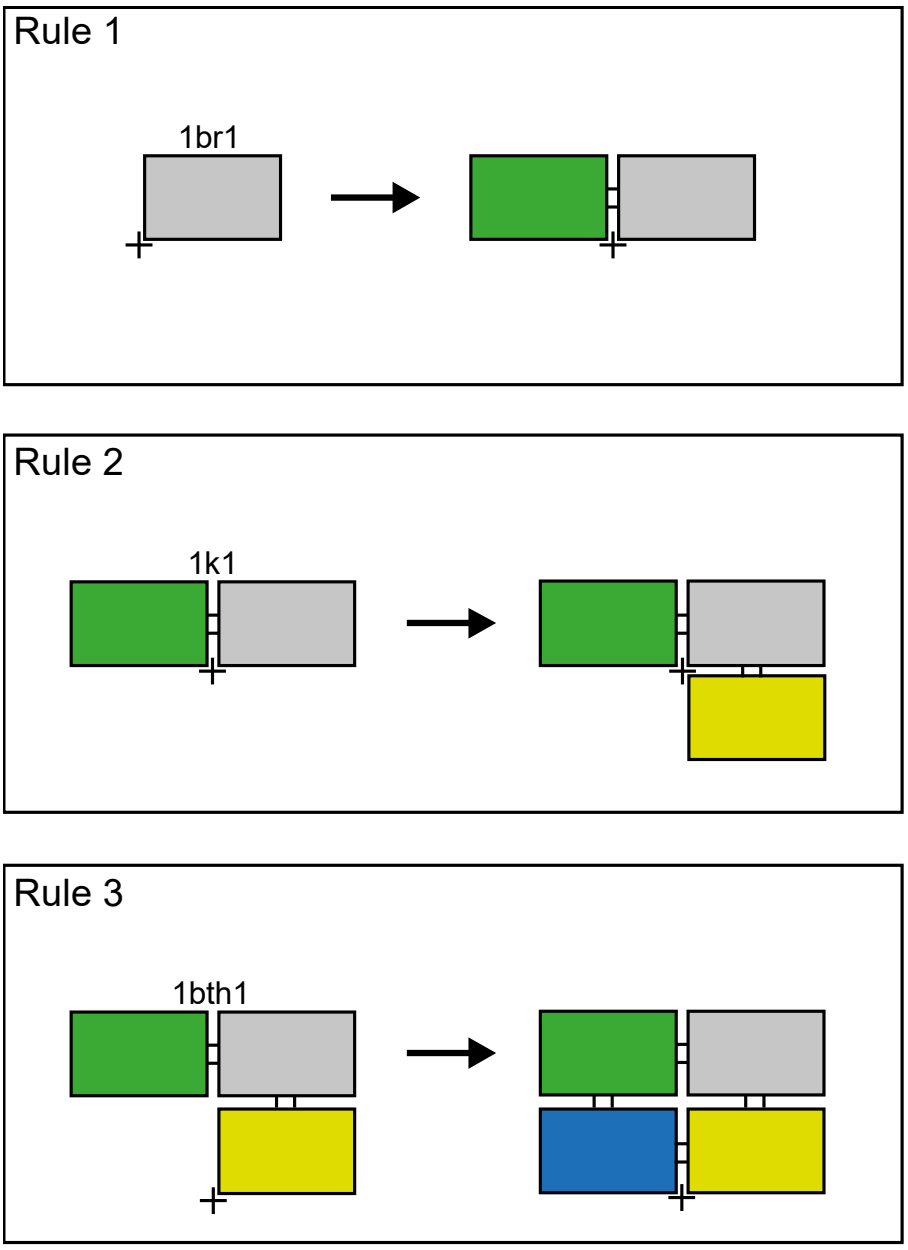


Figure 7.6. Layout rules for the living space

In Rule 1, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= br1 is the design context, which indicates that the layout of a bedroom should be generated based on a certain specification br1. The GDA matches state label sL = br1 if it is the same as the current design goals, represented by the expected function $O_{exp}{}^{F}$ and the expected behaviours $O_{exp}{}^{B}$.

In order to match the state label sL = br1 to the current design goals, the GDA firstly interprets that some persons A_{int} require a bedroom and a layout of the bedroom does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal $O_{exp}{}^{F}$ = br1 (i.e., the layout

of a bedroom is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the bedroom requires the following conditions to be met:

- The layout of the living room is recognised in the VW.
- The design context (br1) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 2, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= k1 is the design context, which indicates that the layout of a kitchen should be generated based on a certain specification k1. The GDA matches state label sL = k1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = k1 to the current design goals, the GDA firstly interprets that some persons A_{int} are indicating that they would require a kitchen and that a layout of the kitchen does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = k1 (i.e., the layout of a kitchen is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the kitchen requires the following conditions to be met:

- The layout of the living room is recognised in the VW.
- The layout of the bedroom is recognised in the VW.
- The design context (k1) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 3, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= bth1 is the design context, which indicates that the layout of a bathroom should be generated based on a certain specification bth1. The GDA matches state label sL = bth1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = bth1 to the current design goals, the GDA firstly interprets that some persons A_{int} require a bathroom and that a layout of the bathroom does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = bth1 (i.e., the layout of a bathroom is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the bathroom requires the following conditions to be met:

- The layout of the living room is recognised in the VW.
- The layout of the bedroom is recognised in the VW.
- The layout of the kitchen is recognised in the VW.
- The design context (bth1) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

7.2.2. Object Rules

The second set of rules that the GDA applies are the GDG's object rules, which are used to configure the living space with various objects that form its visual boundaries. Such objects also provide visual cues for the types of activities that the living space supports, people's notion of the place as a living space and their orientation around it.

Fig. 7.6 shows that after the GDA applies the GDG's object rules for the living room in the living space application, a visual boundary of it is generated. It (Fig. 7.6) also shows that

after the GDA applies the GDG's object rules for the bedroom in the living space application, a visual boundary of it is generated. The same figure (Fig. 7.6) further shows that after the GDA applies the GDG's object rules for the kitchen in the living space application, a visual boundary of it is generated. Finally, the figure (Fig. 7.6) shows that after the GDA applies the GDG's object rules for the bathroom in the living space application, a visual boundary of it is generated.

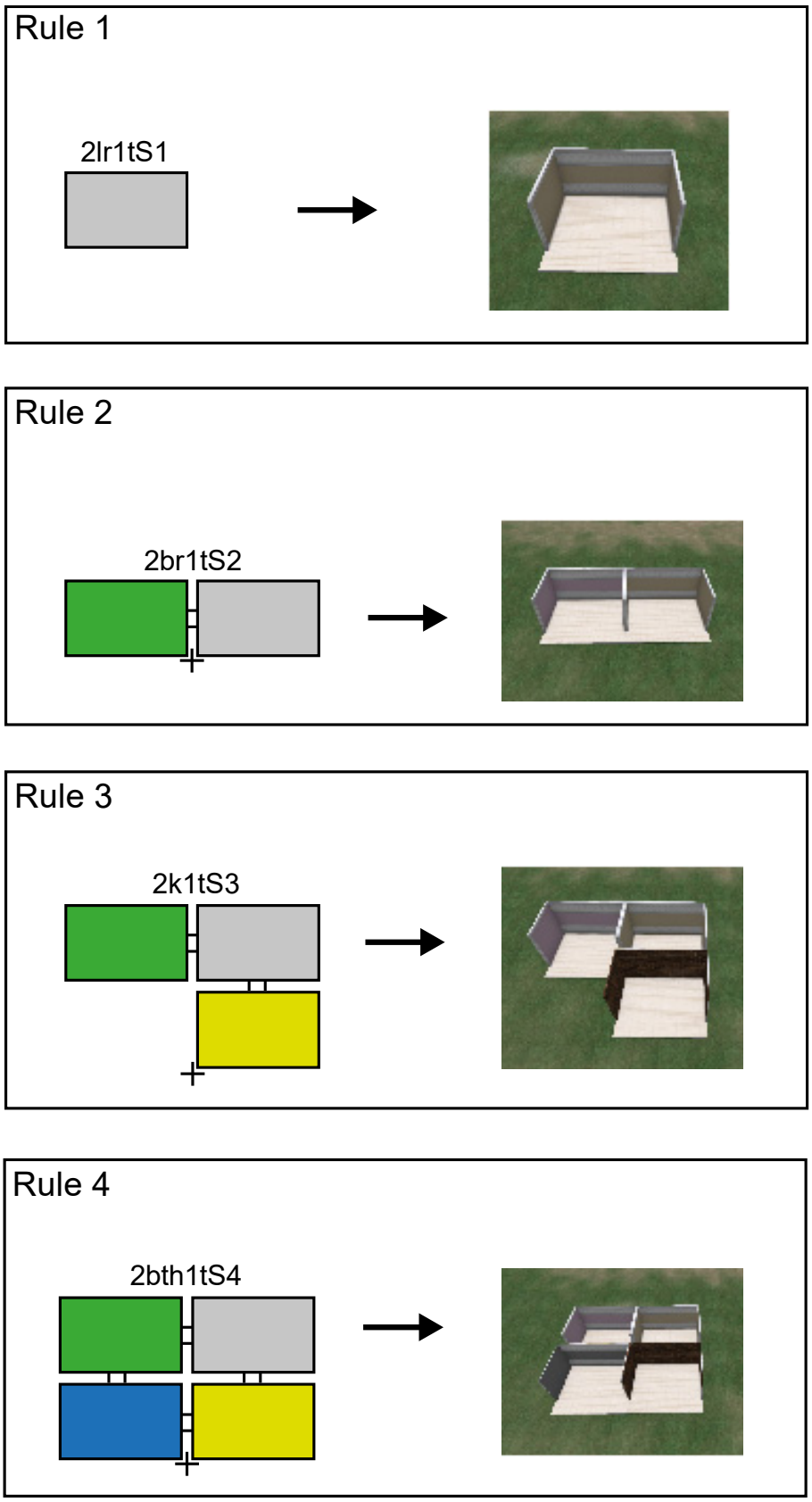


Figure 7.7. Object rules for configuring the visual boundary of the living space

In Rule 1, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= lr1tS1 is the design context, which indicates that a certain texture scheme should be applied to the interior walls of the living room. The GDA

matches state label sL = lr1tS1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = lr1tS1 to the current design goals, the GDA firstly interprets that some persons A_{int} require the interior walls of the office to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = lr1tS1 (i.e., a certain texture scheme for the interior walls of the living room is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 1 for generating a texture scheme for the interior walls of the living room requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The design context (lr1tS1) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 2, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= br1tS2 is the design context, which indicates that a certain texture scheme should be applied to the interior walls of the bedroom. The GDA matches state label sL = br1tS2 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = br1tS2 to the current design goals, the GDA firstly interprets that some persons A_{int} require the interior walls of the bedroom to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = br1tS2 (i.e., a certain texture scheme for the interior walls of the bedroom is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 2 for generating a texture scheme for the interior walls of the bedroom requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The visual boundary of the bedroom is recognised in the VW.
- The design context (br1tS2) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 3, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= k1tS3 is the design context, which indicates that a certain texture scheme should be applied to the interior walls of the kitchen. The GDA matches state label sL = k1tS3 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = k1tS3 to the current design goals, the GDA firstly interprets that some persons A_{int} are indicating that they would require the interior walls of the kitchen to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = k1tS3 (i.e., a certain texture scheme for the interior walls of the kitchen is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 3 for generating a texture scheme for the interior walls of the kitchen requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The visual boundary of the bedroom is recognised in the VW.
- The visual boundary of the kitchen is recognised in the VW.
- The design context (k1tS3) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 4, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= bth1tS4 is the design context, which indicates that a certain texture should be applied to the interior walls of the bathroom. The GDA matches state label sL = bth1tS4 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = bth1tS4 to the current design goals, the GDA firstly interprets that some persons A_{int} require the interior walls of the bathroom to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = bth1tS4 (i.e., a certain texture scheme for the interior walls of the bathroom is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 4 for generating a texture scheme for the interior walls of the bathroom requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The visual boundary of the bedroom is recognised in the VW.
- The visual boundary of the kitchen is recognised in the VW.
- The visual boundary of the bathroom is recognised in the VW.
- The design context (bth1tS4) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

Fig. 7.7 shows that after the GDA applies the GDG's object rules for the living room, it is also configured with visual cues (i.e., a sofa, table, bookcase, etc.) for the types of activities that it supports, people's notion of the place as a living room and their orientation around it. The same figure (Fig. 7.7) shows that after the GDA applies the GDG's object rules for the bedroom, it is configured with visual cues (i.e., a bed, a bedside drawer, a set of nightlamps, etc.) for the types of activities that it supports, people's notion of the place as a bedroom and their orientation around it. The figure (Fig. 7.7) further shows that after the GDA applies

the GDG's object rules for the kitchen, it is configured with visual cues (i.e., a kitchen table, a sink, stove, etc.) for the types of activities that it supports, people's notion of the place as a kitchen and their orientation around it. Finally, the figure (Fig. 7.7) shows that after the GDA applies the GDG's object rules for the bathroom in the living space application, it is configured with visual cues (i.e., a bathtub, a mirror, towel dryer, etc.) for the types of activities that it supports, people's notion of the place as a bathroom and their orientation around it.

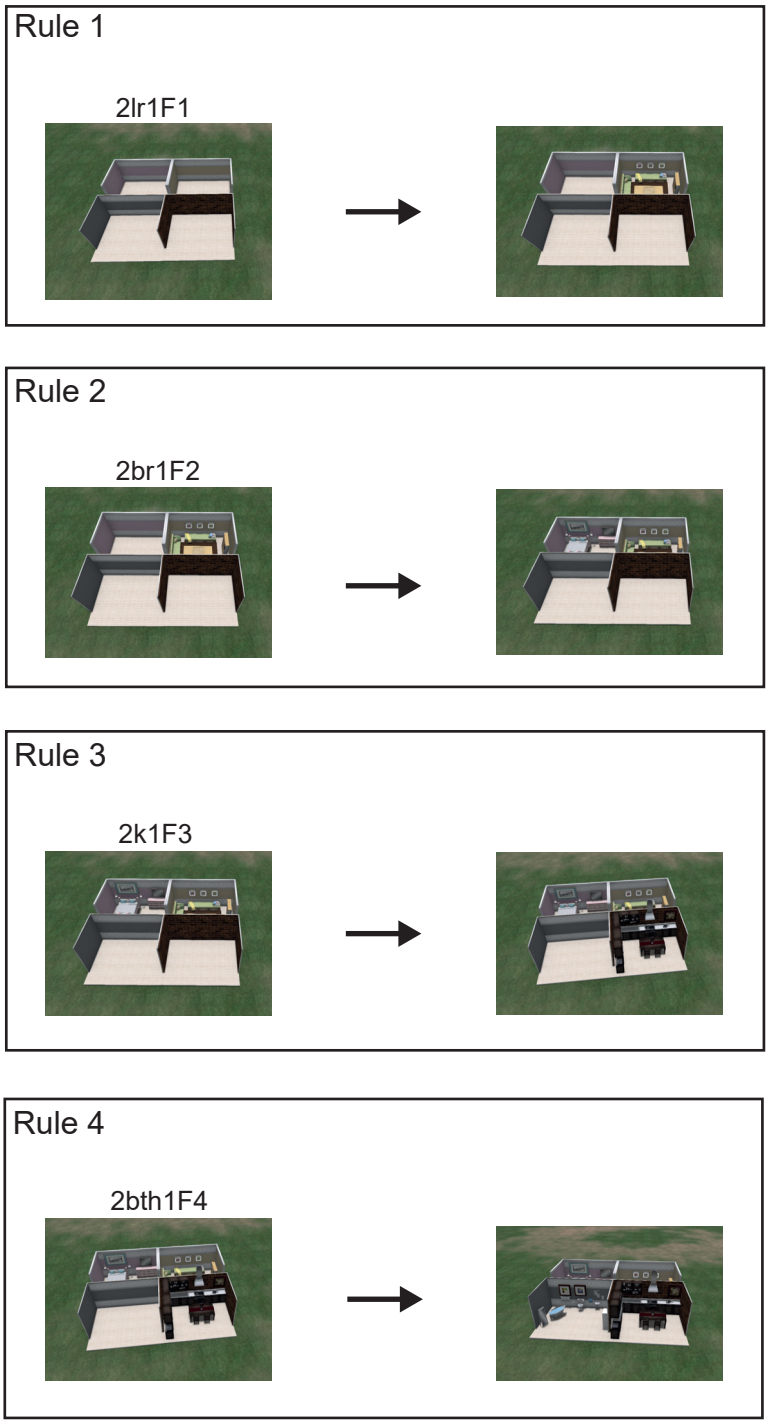


Figure 7.8. Object rules for configuring the living space with visual cues

In Rule 1, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= lr1F1 is the design context, which indicates that a set of furniture should be generated in the living room. The GDA matches state label sL = lr1F1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = lr1F1 to the current design goals, the GDA firstly interprets that some persons A_{int} require a set of furniture in the living room and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = lr1F1 (i.e., a set of furniture in the living room is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 1 for generating a set of furniture in the living room requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The design context (lr1F1) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 2, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= br1F2 is the design context, which indicates that a set of furniture should be generated in the bedroom. The GDA matches state label sL = br1F2 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = br1F2 to the current design goals, the GDA firstly interprets that some persons A_{int} require a set of furniture in the bedroom and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = br1F2 (i.e., a set of furniture in the bedroom is needed). Lastly, the GDA selects a design

rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 2 for generating a set of furniture in bedroom requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The visual boundary of the bedroom is recognised in the VW.
- The design context (br1F2) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 3, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= k1F3 is the design context, which indicates that a set of furniture should be generated on the kitchen. The GDA matches state label sL = k1F3 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = k1F3 to the current design goals, the GDA firstly interprets that some persons A_{int} require a set of furniture to be generated in the kitchen and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = k1F3 (i.e., a set of furniture in the kitchen is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 3 for generating a set of furniture in the kitchen requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The visual boundary of the bedroom is recognised in the VW.
- The visual boundary of the kitchen is recognised in the VW.

- The design context (k1F3) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

In Rule 4, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= bth1F4 is the design context, which indicates that a set of furniture should be generated in the bathroom. The GDA matches state label sL = bth1F4 if it is the same as the current design goals, represented by the expected function $O_{exp}{}^{F}$ and the expected behaviours $O_{exp}{}^{B}$.

In order to match the state label sL = bth1F4 to the current design goals, the GDA firstly interprets that some persons A_{int} require a set of furniture in the bathroom and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal $O_{exp}{}^{F}$ = bth1F4 (i.e., a set of furniture in the bathroom is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 4 for generating a set of furniture in the bathroom requires the following conditions to be met:

- The visual boundary of the living room is recognised in the VW.
- The visual boundary of the bedroom is recognised in the VW.
- The visual boundary of the kitchen is recognised in the VW.
- The visual boundary of the bathroom is recognised in the VW.
- The design context (bth1F4) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

7.2.3. Navigation Rules

The third set of rules that the GDA applies are the navigation rules, which are used to configure the living space with wayfinding aids such as hyperlinks and teleportation devices.

Similar to the previous example, the close proximity of the rooms in the living space makes it unnecessary to implement wayfinding aids. However, Fig. 7.8 shows a possible navigation rule for teleportation between all the rooms in the living space.

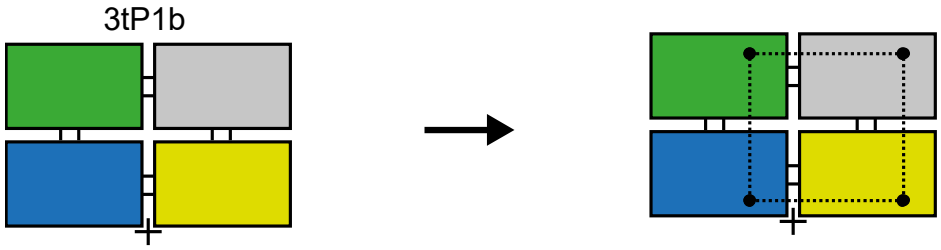


Figure 7.9. Navigation rule that establishes teleportation in the living space

After applying this navigation rule, the RHO (i.e., the model of the living space with teleportation enabled between all of its rooms) will replace the LHO (i.e., the model of the living space without teleportation enabled between any of its rooms).

The state label sL = 3 shows that we are working in the third phase of the GDG's design rules and sL = tP1b is the design context, which indicates that teleportation devices should be generated in order to allow movement between all of the rooms.

Applying the navigation rule that configures the living space with way finding aids requires the following conditions to be met:

- Visual boundaries of all the rooms (i.e., the living room, bedroom, kitchen and bathroom) have been generated.
- The rooms (i.e., the living room, bedroom, kitchen and bathroom) have been configured with all necessary visual cues.
- The design context (tP1b) matches some current design requirements or needs that are supposed to be used by the GDA to model the living space.

7.2.4. Interaction Rules

The fourth set of rules that the GDA applies are interaction rules, which are used for designing algorithms, writing code and ascribing scripts to objects to enable users to be

able to interact with the living space. The following are the IF…THEN… statements that could be used to implement teleportation in the living space application.

sL = 4

IF: Visual boundaries of all the rooms have been generated

AND

All the rooms have been configured with all necessary visual cues

THEN: render teleport devices in all of the rooms

AND

Ascribe appropriate behaviour to teleport devices according to design requirements

Fig. 7.10 shows the final design of the living space, which was creating using the rules (i.e., layout design rules, object design rules, navigation design rules and interaction design rules) that have been described in the preceding sections (i.e., Sections 7.2.1 to 7.2.4).

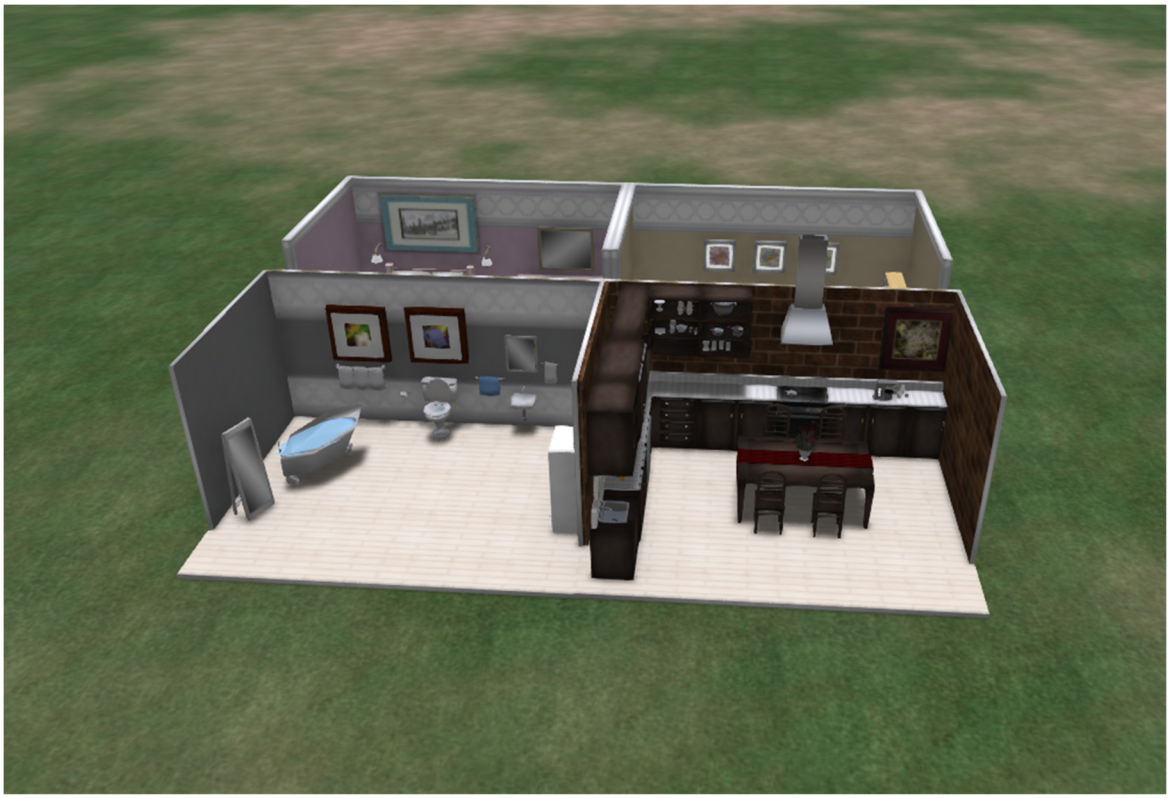
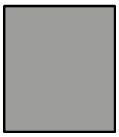
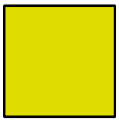


Figure 7.10. The final design of the living space

7.3. Designing a Student Centre Application

For the purpose of designing the student centre application, the following symbols and their meanings (Table 7.3) will be used:

Table 7.3. A summary of the symbols used in the student centre application and their meanings

Symbol	Meaning
	The main floor of a library
	The top floor of the library (used as a quiet study space)
	A set of stairs that lead into the library
	A union shop
	A postgraduate centre
	A restaurant

7.3.1. Layout Rules

The first set of rules that the GDA applies are the GDG's layout rules, which are used to create the layout of the student centre.

Fig. 7.9 shows five sets of layout rules that the GDA uses to create the layout of the student centre. After the application of the first set of layout rules, the RHO (i.e., the layout of the main and top floors of the library) will replace the LHO (i.e., the layout of the main floor of the library). After the application of the second set of layout rules, the RHO (i.e., the layout of the main and top floors of the library with stairs in the front) will replace the LHO (i.e., the layout of the main and top floors of the library). After the application of the third set of layout rules, the RHO (i.e., the layout of the library and union shop) will replace the LHO (i.e., the layout of the main and top floors of the library with stairs in the front). After the application

of the fourth set of layout rules, the RHO (the layout of the library, union shop and the postgraduate centre) will replace the LHO (the layout of the library and union shop). After the application of the fifth set of layout rules, the RHO (the layout of the library, union shop, postgraduate centre and restaurant) will replace the LHO (the layout of the library, union shop and the postgraduate centre).

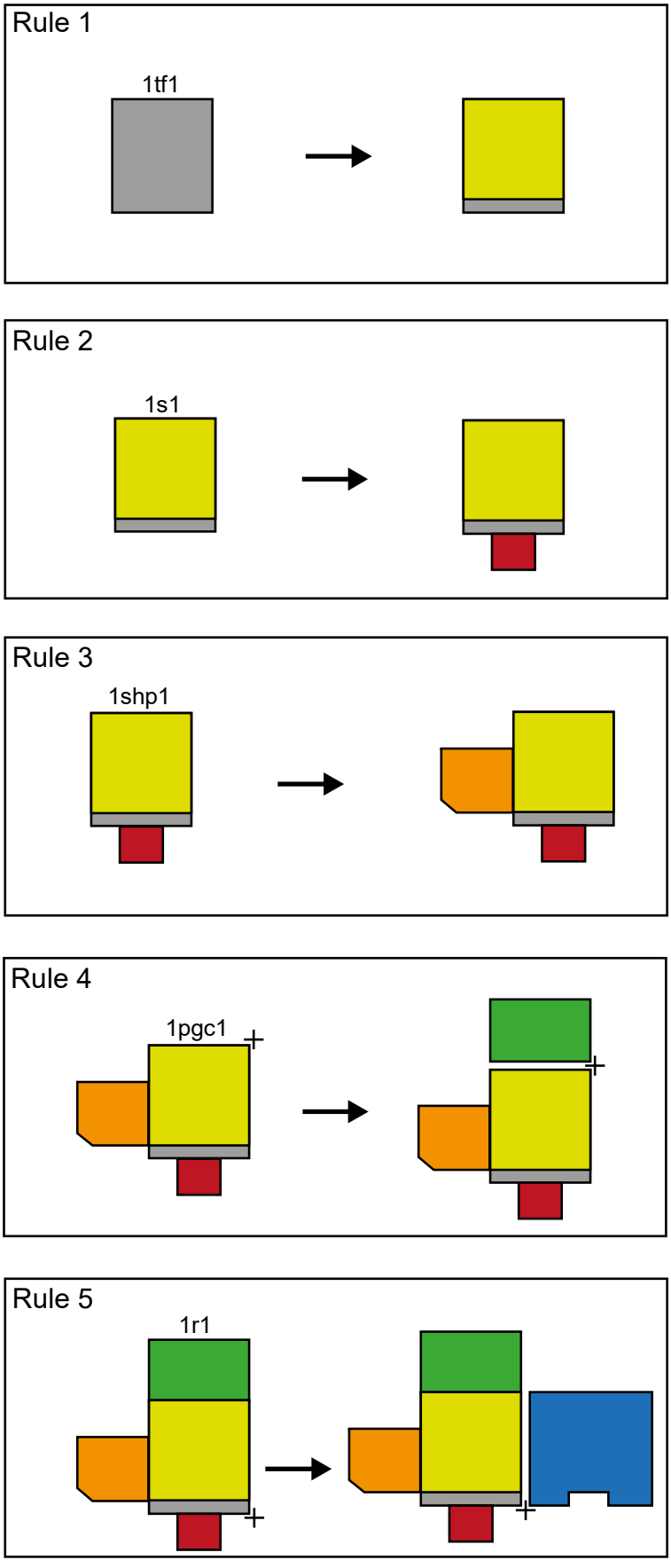


Figure 7.11. Layout rules for the student centre

In Rule 1, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= tf1 is the design context, which indicates that the layout of the top floor of the library should be generated based on a certain specification tf1. The GDA matches state label sL = tf1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = tf1 to the current design goals, the GDA firstly interprets that some students A_{int} require a quiet study area and a layout of the top floor of the library does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = tf1 (i.e., the layout of the top floor of the library is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the top floor of the library requires the following conditions to be met:

- The layout of the main floor of the library is recognised in the VW.
- The design context (tf1) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 2, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= s1 is the design context, which indicates that the layout of some stairs should be generated based on a certain specification s1. The GDA matches state label sL = s1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = s1 to the current design goals, the GDA firstly interprets that some persons A_{int} may wish to enter the library and the layout of stairs that lead into it does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = s1 (i.e., the layout of some stairs that lead into the library is needed). Lastly, the GDA selects a design

rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the stairs requires the following conditions to be met:

- The layout of the main floor of the library is recognised in the VW.
- The layout of the top floor of the library is recognised in the VW.
- The design context (s1) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 3, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= shp1 is the design context, which indicates that the layout of a shop should be generated based on a certain specification shp1. The GDA matches state label sL = shp1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = shp1 to the current design goals, the GDA firstly interprets that some persons A_{int} would require a shop and a layout of the shop does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = shp1 (i.e., the layout of a shop is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the shop requires the following conditions to be met:

- The layout of the main floor of the library is recognised in the VW.
- The layout of the top floor of the library is recognised in the VW.
- The layout of the stairs that lead into the library is recognised in the VW.
- The design context (shp1) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 4, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= pgc1 is the design context, which indicates that the layout of a postgraduate centre should be generated based on a certain specification pgc1. The GDA matches state label sL = pgc1 if it is the same as the current design goals, represented by the expected function $O_{exp}{}^{F}$ and the expected behaviours $O_{exp}{}^{B}$.

In order to match the state label sL = pgc1 to the current design goals, the GDA firstly interprets that some masters and PhD students A_{int} require a postgraduate centre and a layout of the postgraduate centre does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal $O_{exp}{}^{F}$ = pgc1 (i.e., the layout of a postgraduate centre is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the postgraduate centre requires the following conditions to be met:

- The layout of the main floor of the library is recognised in the VW.
- The layout of the top floor of the library is recognised in the VW.
- The layout of the stairs that lead into the library is recognised in the VW.
- The layout of the union shop is recognised in the VW.
- The design context (pgc1) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 5, the state label sL = 1 shows that the GDA is in the process of applying the GDG's first level of design rules and sL= r1 is the design context, which indicates that the layout of a restaurant should be generated based on a certain specification r1. The GDA matches state label sL = r1 if it is the same as the current design goals, represented by the expected function $O_{exp}{}^{F}$ and the expected behaviours $O_{exp}{}^{B}$.

In order to match the state label sL = r1 to the current design goals, the GDA firstly interprets that some persons A_{int} require a restaurant and a layout of the restaurant does not yet exist O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = r1 (i.e., the layout of a restaurant is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the layout rule for generating the layout of the restaurant requires the following conditions to be met:

- The layout of the main floor of the library is recognised in the VW.
- The layout of the top floor of the library is recognised in the VW.
- The layout of the stairs that lead into the library is recognised in the VW.
- The layout of the union shop is recognised in the VW.
- The layout of the postgraduate centre is recognised in the VW.
- The design context (r1) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

7.3.2. Object Rules

The second set of rules that the GDA applies are the GDG's object rules, which are used to configure the student centre with various objects that form its visual boundaries. Such objects also provide visual cues for the types of activities that the student centre supports, people's notion of the place as a student centre and their orientation around it.

Fig. 7.10 shows six sets of object rules that the GDA applies to create the visual boundaries (and visual cues) of the student centre. After the application of the first set of object rules, a visual boundary of the main floor of the library is generated. After the application of the second set of object rules, a visual boundary of the top floor of the library is generated. After the application of the third set of object rules, a visual boundary of the stairs that lead into the library is generated. After the application of the fourth set of object rules, a visual boundary of the union shop is generated. After the application of the fifth set of object rules,

a visual boundary of the postgraduate centre is generated. Finally, after the application of the sixth set of object rules, a visual boundary of the restaurant (and essentially the entire application) is generated.

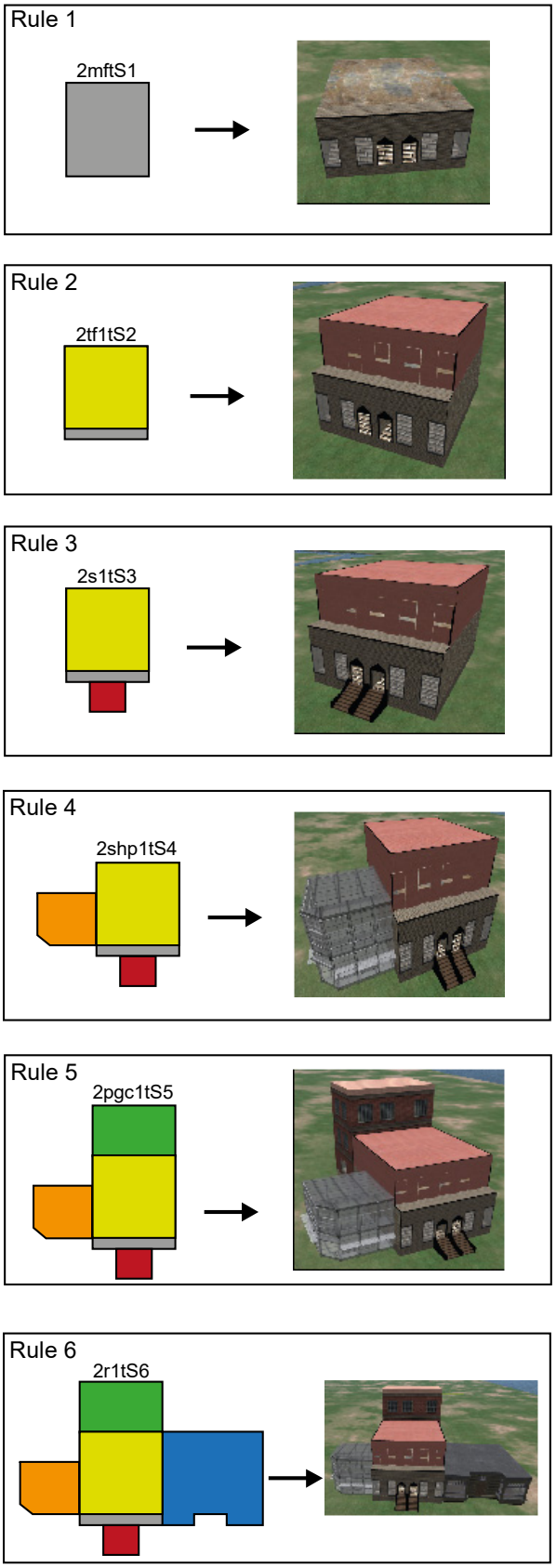


Figure 7.12. Object rules for configuring the visual boundaries of the student centre

In Rule 1, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= mftS1 is the design context, which indicates that a certain texture scheme should be applied to the main floor of the library. The GDA matches state label sL = mftS1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = mftS1 to the current design goals, the GDA firstly interprets that some persons A_{int} require the main floor of the library to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = mftS1 (i.e., a certain texture scheme for the main floor of the library is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 1 for generating a texture scheme for the main floor of the library requires the following conditions to be met:

- The visual boundary of the main floor of the library is recognised in the VW.
- The design context (mftS1) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 2, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= tf1tS2 is the design context, which indicates that a certain texture scheme should be applied to the top floor of the library. The GDA matches state label sL = tf1tS2 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = tf1tS2 to the current design goals, the GDA firstly interprets that some persons A_{int} require the top floor of the library to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = tf1tS2 (i.e., a certain texture scheme for the top floor of the library is

needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 2 for generating a texture scheme for the top floor of the library requires the following conditions to be met:

- The visual boundary of the main floor of the library is recognised in the VW.
- The visual boundary of the top floor of the library is recognised in the VW.
- The design context (tf1tS2) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 3, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= s1tS3 is the design context, which indicates that a certain texture scheme should be applied to the stairs that lead into the library. The GDA matches state label sL = s1tS3 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = s1tS3 to the current design goals, the GDA firstly interprets that some persons A_{int} require the stairs that lead into the library to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = s1tS3 (i.e., a certain texture scheme for the stairs that lead into the library is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 3 for generating a texture scheme for the stairs that lead into the library requires the following conditions to be met:

- The visual boundary of the main floor of the library is recognised in the VW.
- The visual boundary of the top floor of the library is recognised in the VW.
- The visual boundary of the stairs that lead into the library is recognised in the VW.

- The design context (s1tS3) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 4, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= shp1tS4 is the design context, which indicates that a certain texture should be applied to the union shop. The GDA matches state label sL = shp1tS4 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = shp1tS4 to the current design goals, the GDA firstly interprets that some persons A_{int} require the union shop to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = shp1tS4 (i.e., a certain texture scheme for the union shop is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 4 for generating a texture scheme for the union shop requires the following conditions to be met:

- The visual boundary of the main floor of the library is recognised in the VW.
- The visual boundary of the top floor of the library is recognised in the VW.
- The visual boundary of the stairs that lead into the library is recognised in the VW.
- The visual boundary of the union shop is recognised in the VW.
- The design context (shp1tS4) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 5, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= pgc1tS5 is the design context, which indicates that a certain texture should be applied to the postgraduate centre. The GDA matches state label

sL = pgc1tS5 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = pgc1tS5 to the current design goals, the GDA firstly interprets that some persons A_{int} require the postgraduate centre to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = pgc1tS5 (i.e., a certain texture scheme for the postgraduate centre is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 5 for generating a texture scheme for the postgraduate centre requires the following conditions to be met:

- The visual boundary of the main floor of the library is recognised in the VW.
- The visual boundary of the top floor of the library is recognised in the VW.
- The visual boundary of the stairs that lead into the library is recognised in the VW.
- The visual boundary of the union shop is recognised in the VW.
- The visual boundary of the postgraduate centre is recognised in the VW.
- The design context (pgc1tS5) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

In Rule 6, the state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= r1tS6 is the design context, which indicates that a certain texture should be applied to the restaurant. The GDA matches state label sL = r1tS6 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = r1tS6 to the current design goals, the GDA firstly interprets that some persons A_{int} require the postgraduate centre to have a certain texture and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the

design goal O_{exp}^{F} = r1tS6 (i.e., a certain texture scheme for the postgraduate centre is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying Rule 6 for generating a texture scheme for the postgraduate centre requires the following conditions to be met:

- The visual boundary of the main floor of the library is recognised in the VW.
- The visual boundary of the top floor of the library is recognised in the VW.
- The visual boundary of the stairs that lead into the library is recognised in the VW.
- The visual boundary of the union shop is recognised in the VW.
- The visual boundary of the postgraduate centre is recognised in the VW.
- The visual boundary of the restaurant is recognised in the VW.
- The design context (r1tS6) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

Fig 7.11 shows some objects that are part of the furniture that is used to provide visual cues for the restaurant in the student centre.

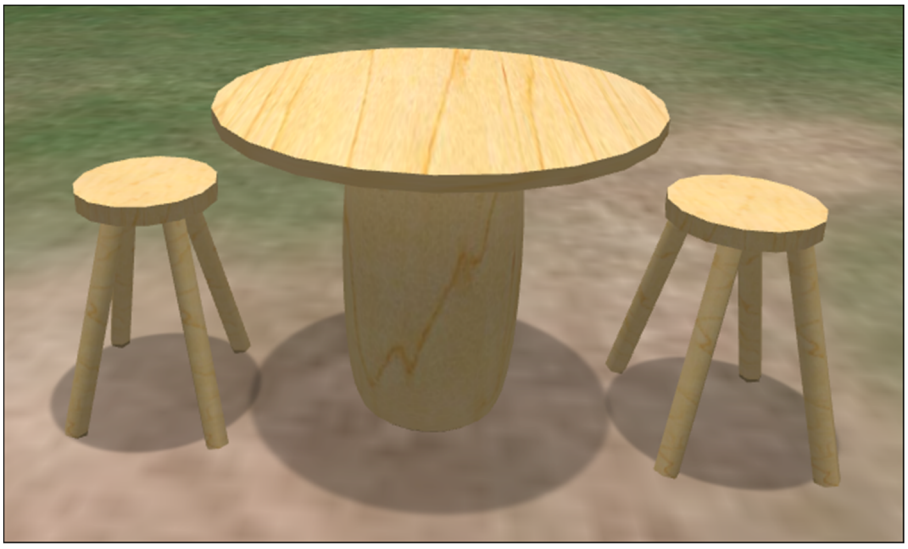


Figure 7.13. A set of objects that are used to provide visual cues for the restaurant in the student centre

Fig. 7.12 shows that after the GDA applies the GDG's object rules for the restaurant in the student centre, it is configured with visual cues (e.g., stools and tables) for the types of activities that it supports, people's notion of the place as a restaurant and their orientation around it.

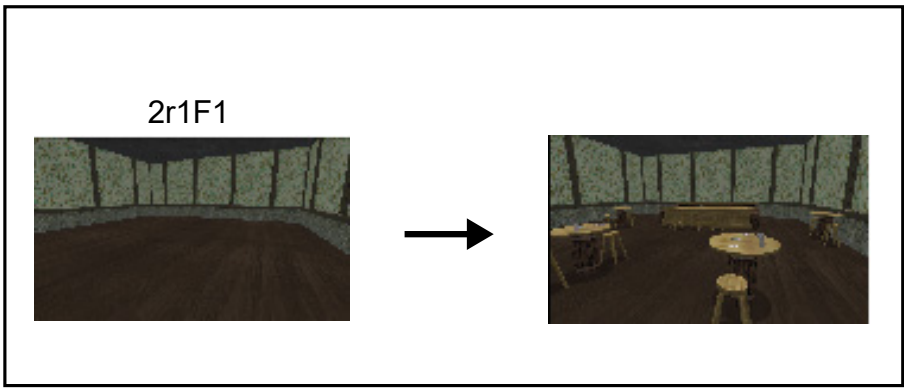


Figure 7.14. Object rule for configuring the restaurant in the student centre with visual cues

The state label sL = 2 shows that the GDA is in the process of applying the GDG's second level of design rules and sL= r1F1 is the design context, which indicates that a set of furniture should be generated in the restaurant. The GDA matches state label sL = r1F1 if it is the same as the current design goals, represented by the expected function O_{exp}^{F} and the expected behaviours O_{exp}^{B}.

In order to match the state label sL = r1F1 to the current design goals, the GDA firstly interprets that some persons A_{int} require a set of furniture in the restaurant and that this condition has not yet been met O_{int}. Secondly, the GDA hypothesises the design goal O_{exp}^{F} = r1F1 (i.e., a set of furniture in the restaurant is needed). Lastly, the GDA selects a design rule to be applied only if the LHO of the rule is found and its state label matches the design goal.

Applying the object rule for generating a set of furniture in the restaurant of the student centre requires the following conditions to be met:

- The visual boundary of the main floor of the library is recognised in the VW.
- The visual boundary of the top floor of the library is recognised in the VW.
- The visual boundary of the stairs that lead into the library is recognised in the VW.

- The visual boundary of the union shop is recognised in the VW.
- The visual boundary of the postgraduate centre is recognised in the VW.
- The visual boundary of the restaurant is recognised in the VW.
- The design context (r1F1) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

7.3.3. Navigation Rules

The third set of rules that the GDA applies are the navigation rules, which are used to configure the student centre with wayfinding aids such as hyperlinks and teleportation devices.

Fig. 7.13 shows a navigation rule that implements wayfinding aids for the student centre.

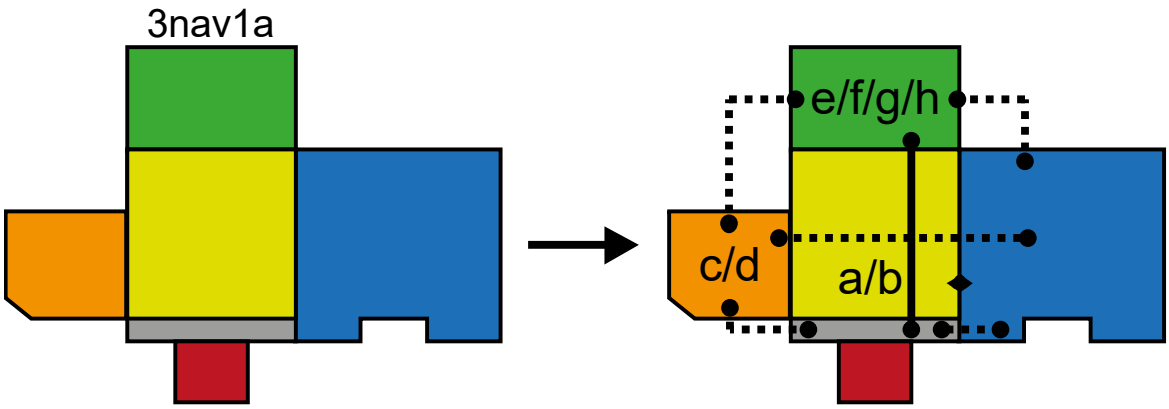


Figure 7.15. Navigation rule that establishes a series of wayfinding aids for the student centre

After applying this navigation rule, the RHO (i.e., the model of the student centre with elements of navigation enabled between all of its buildings and rooms) will replace the LHO (i.e., the model of the living space without any elements of navigation).

The state label sL = 3 shows that the GDA is in the process of applying the GDG's third level of design rules and sL = nav1a is the design context, which indicates that certain elements of navigation should be generated in order to allow movement (or make it easier to move) between all of the buildings and rooms in the student centre.

Applying the navigation rule for enabling elements of navigation between the buildings and rooms of the student centre requires the following conditions to be met:

- Visual boundaries of all the buildings and rooms have been generated.
- The building and rooms have been configured with all necessary visual cues.
- The design context (nav1a) matches some current design requirements or needs that are supposed to be used by the GDA to model the student centre.

7.3.4. Interaction Rules

The fourth set of rules that the GDA applies are interaction rules, which are used for designing algorithms, writing code and ascribing scripts to objects to enable users to be able to interact with the VW application. The following are the IF…THEN… statements that could be used to implement a greeter that offers a menu to people who visit the restaurant in the student centre application.

sL = 4

IF: The visual boundary of the restaurant has been generated

AND

IF: The restaurant has been configured with all necessary visual cues

AND

IF: A visitor is sensed in the restaurant

AND

The visitor has not been offered the current menu

THEN: Greet the visitor

AND

Offer the visitor the current menu

Fig. 7.16 shows the final design of the student centre, which was creating using the rules (i.e., layout design rules, object design rules, navigation design rules and interaction design rules) that have been described in the preceding sections (i.e., Sections 7.3.1 to 7.3.4).

Figure 7.16. The final design of the student centre

7.4. Review of the Functionality of the VWADM

The VWADM is a generative method. Therefore, the case studies focus on showing how GDAs can be used to dynamically generate VW applications by interpreting and applying the design rules that pertain to GDGs. However, there are a number of scenarios in which a specification or model may no longer be required. For example, a meeting room may no longer be needed after a meeting (i.e., it won't be useful until the next time it is required). In such a case, it could be demolished in order to save disk space, memory and other computing resources.

Based on this idea, the GDG framework accounts for two types of rules - additive and subtractive rules - that the GDA is capable of reading, interpreting and executing. Additive rules are used for generating specifications or models of VW applications. In complement, subtractive rules are used for changing or demolishing them. Fig. 7.14 shows an example of a subtractive layout rule to remove the meeting room from the office application.

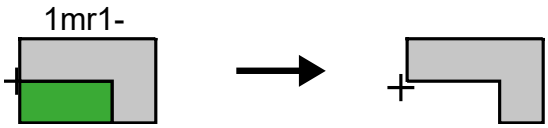


Figure 7.17. Layout rule for removing the meeting room from the office application

Similar to additive rules, subtractive rules can subsequently be applied to a VW application's layout design, object design, navigation design and interaction design.

Another point worth mentioning is that besides the number that indicates the level of the design phase (i.e., layout design = 1, object design = 2, navigation design = 3 and interaction design = 4), the naming of state labels is arbitrary and has no conventional rules. However, as per programming conventions it is useful to give them names that are indicative to some degree about their design context or the role they play in the design process. As such, it is up to the designer to define the meanings of state labels. In terms of their use by GDA's, state labels can be coded as functions or states that a GDA can selectively read, interpret and execute.

7.5. Summary

This chapter presented three case studies that were used to demonstrate the functionality of the VWADM. In the first case study, the VWADM was used to design an office application. In the second case study, it was used to design a living space application. The VWADM was used in the third case study to design a student centre application. Finally, in light of the case studies, a brief review was given of some extended features and capabilities of the VWADM.

8. Evaluation of the VWADM

This chapter discusses a user study that was conducted to evaluate the features and capabilities of the VWADM with respect to their significance for practical application in designing VW applications. As mentioned in the Aims and Objectives (Section 1.3), the VWADM is intended for use by developers to enable them to create place models of VW applications during the design process. Therefore, one aim of the study was to obtain their opinions about its fitness and utility for doing so. The chapter begins by providing an overview of the fitness-utility model, which was used to design the study and formulate the criteria (i.e., fitness and utility) for evaluation. Next, it provides a brief overview of the methods and instruments that were chosen for capturing the opinions of developers during the study. This is followed by a statement on the preparatory activities that were performed to recruit developers to participate in the study. The statement on the preparatory activities of the study is followed by discussions of some ethical considerations and the procedure that was used to conduct the study. These two sections are followed by information from the study, which include participant experience information, a report on the reliability of the questionnaire used in the study and the results of the study. The chapter finishes with a discussion of the results of the study.

8.1. The Fitness-Utility Model

The design of the study is based on the fitness-utility model (Gill & Hevner, 2011), which is an evaluation model for capturing the evolutionary nature of artefacts and the essential DSR nature of searching for a satisfactory solution across a design landscape. Current thinking in DSR defines the usefulness of an artefact as the primary research goal (Mustafa, et al., 2014). However, the fitness-utility model proposes that evaluation needs to move from measures of usefulness alone to utility functions that incorporate the evolution and survival

of the artefact in its design landscape[7]. This view suggests usefulness as one out of several other (mostly fitness) characteristics for evaluating an artefact that include decomposability, malleability, openness, embedment in a design system, novelty, interestingness and elegance. Fig. 8.1 shows the fitness-utility model.

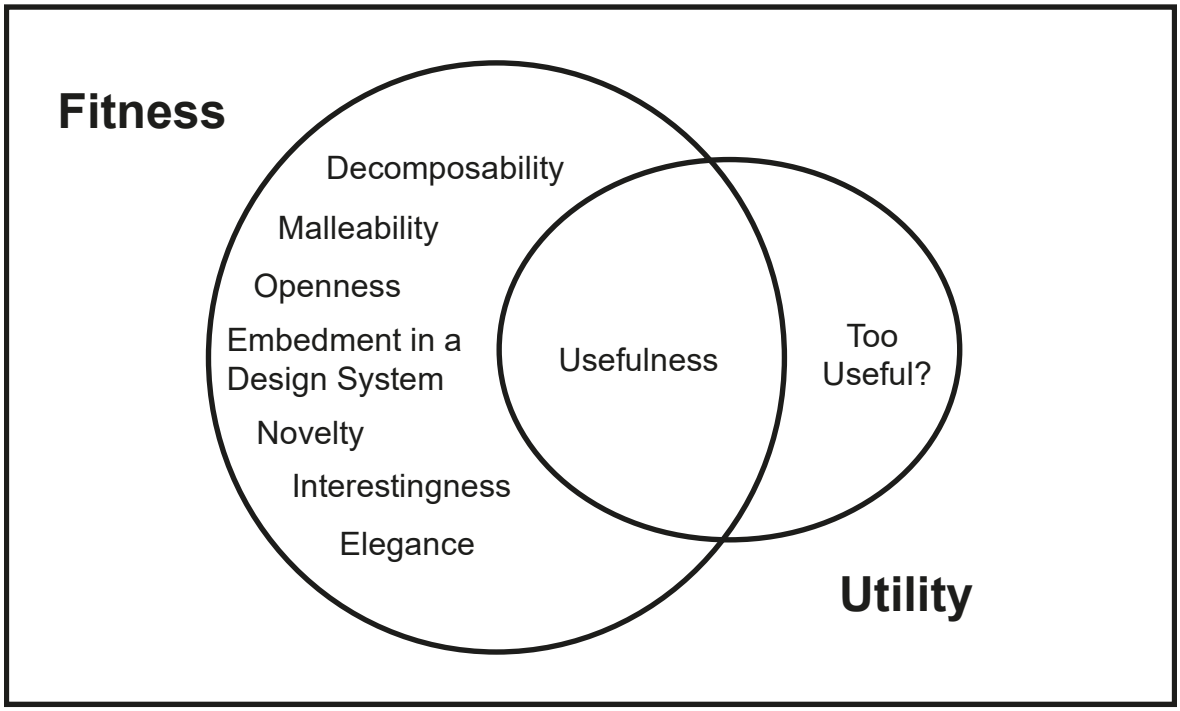


Figure 8.1. The Fitness-Utility Model (Gill & Hevner, 2013)

The characteristics of the fitness-utility model are defined as follows:

Decomposability: is an artefact's ability to evolve from decomposable (or nearly decomposable) subsystems. On one hand, if an artefact cannot be decomposed into an independent (or nearly independent) subsystem, its evolution would tend to be inflexible (Simon, 1962). On the other hand, if the artefact is built on separable components, then some of those components may exhibit high levels of fitness and evolve rapidly while the others may remain static or be discarded.

[7] The rationale for this is that together, fitness and utility provide a set of characteristics for evaluating an artefact that are related to its long-term value in its design landscape, as opposed to its immediate usefulness only.

Malleability: is the degree to which an artefact can be adapted by its users and respond to changes in its environment (e.g., changes in its use or its market).

Openness: is the degree to which an artefact is open to inspection, modification and reuse. This characteristic tends to encourage the evolution of an artefact by making it easy to see how an it is designed and to modify its components.

Embedment in a design system: refers to the origin of the artefact (as per its evolution). It is expected that an artefact will evolve more rapidly if it is the product of an environment in which design is routinely practiced as opposed to one that is produced in a context where design is an unusual activity.

Novelty: an artefact may be considered novel if it originates from an entirely new region of the design landscape.

Interestingness: if the purpose for creating an artefact is to explore or demonstrate some specific purpose, then its interestingness is a demonstration of any emergent or unexpected behaviour that is worthy of subsequent investigation.

Elegance: a crucial concern in design is form versus function - form is an aesthetic concern and function is a practical one. In DSR, elegance is a concept that corresponds to form.

Usefulness: is the degree to which a person believes that the use of an artefact would enhance his or her job performance[8].

[8] This definition of usefulness makes it a proxy for perceived usefulness as described in (Davis, 1989) Technology Acceptance Model (TAM).

8.2. Methods and Instruments

A survey was used as the method for conducting the study. In particular, a questionnaire was developed and hosted online to collect data on the opinions of developers about the VWADM's fitness and utility for designing VW applications (see Appendix E)[9]. The questionnaire was comprised of two sections, each of which contained two items. The items in the first section are a set of two instructions. The first instruction solicited the opinions of the participants of the study about the VWADM's fitness for creating place models of VW applications during the design process. The second instruction solicited the opinions of the participants of the study about the VWADM's utility for the same purpose. The items in the second section of the questionnaire are two questions whose purpose was for collecting some data about the experiences of the participants of the study in designing and developing content for VWs. Table 8.1 is a summary of the design of the questionnaire.

Table 8.1. A summary of the design of the questionnaire

	Item number	Type	Designation
Section1	1	Instruction	Scale
	2	Instruction	Scale
Section 2	3	Question	Multiple answer (single choice)
	4	Question	Multiple answer (multiple choice)

Each of the items (numbered 1 to 4) on the questionnaire included sub-items. The sub-items in the first section of the questionnaire are statements that represent the evaluation characteristics of the fitness-utility model and were to be rated on a five-point Likert scale by the participants of the study. The measures on the Likert scale are as follows: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree and Strongly Disagree. Table 8.2 shows a summary of the evaluation statements.

Table 8.2. A summary of the evaluation statements on the questionnaire

Criteria	Statements (Evaluation Characteristics)
Fitness	The VWADM is made up of components that may be considered independent (or nearly independent) from each other, but work together during the design process

[9] The questionnaire was validated using the literature that pertains to both the TAM (for the usefulness characteristic) and Fitness-Utility model.

	The VWADM is AT LEAST ONE of the following: (1) adaptable, (2) extensible or (3) customisable
	The VWADM is AT LEAST ONE of the following: (1) open to inspection (2) open to modification or (3) open to reuse
	The VWADM originates from (or is inspired by) a field or environment in which design (i.e., software or non-software design) is routinely practiced
	The VWADM originates from (or is inspired by) a field that is different from software engineering
	The VWADM is interesting
	AT LEAST ONE of the following can be said about the VWADM: (1) it is compact, (2) it is simple, (2) its use is transparent, (3) its behaviour is transparent or (5) it has clarity of representation
Utility	Using the VWADM would enable developers to create place models of these applications more quickly during design
	Using the VWADM would improve the performance of developers in creating place models of these applications during design
	Using the VWADM would increase the productivity of developers when creating place models of these applications during design
	Using the VWADM would enhance the effectiveness of developers in creating place models of these applications during design
	Using the VWADM would make it easier for developers to create place models of these applications during design
	Developers would find the VWADM useful for creating place models of VW applications during design

The first set of sub-items in the second section of the questionnaire are multiple answers (allowing a single choice only) that were expected to give some insight into the number of years of experience that participants of the study have in designing and developing content for VWs. The second set of sub-items in the second section of the questionnaire are multiple answers (allowing multiple choices) that were expected to provide some information about the VW platforms that participants of the study are experienced in using to design and develop content for VWs. Table 8.3 shows a summary of the evaluation answers.

Table 8.3. A summary of the evaluation answers on the questionnaire

Experience	
Years	VW Platforms
Less than 1 year 1 year or more, but less than 5 years 5 years or more, but less than 10 years 10 years or more, but less than 15 years 15 years or more	Active Worlds Second Life Open Simulator High Fidelity Meshmoon Sansar Other (Not Listed)

8.3. Preparation

In order to recruit developers to participate in the study, a series of discussions about the study (and the research) were held with members of technical groups in Second Life and Sansar and in online forums (SL Universe and Sansar's community on Discord) that are related to these platforms. A total of 50 developers were informed about the plans for the study and given a set of links to some files associated with it[10]. The files included the following:

- A participant information sheet
- A participant consent form
- The instructions for the study
- A summary of the features and capabilities of the VWADM
- The basic user guide for the VWADM's AW agent package
- The advanced user guide for the VWADM's AW agent package
- The source code for the VWADM
- The questionnaire for the study

The developers were instructed to read the participant information sheet and return the signed consent form prior to participating in the study. After this, to begin the study, they were required to read and follow the instructions for it. The general set of instructions to the participants of the study was for them to provide their opinions (by completing the questionnaire) about the VWADM's fitness and utility for creating place models of VW applications based on the following aspects: (1) discussions held with them about the goals of the research (2) a review of the features and capabilities of the VWADM, (3) a review of

[10] It is quite difficult to conduct a survey and get responses from the entire population of interest (Nulty, 2008). Therefore, a non-probability convenience sampling technique was used to shortlist the 50 developers who were solicited to participate in the study.

its user guides and (4) analysis of its source code. The instruction sheet contained links to all the files that were to be reviewed during the study.

8.4. Ethical Considerations

The study was designed and conducted in accordance with Anglia Ruskin's general policy on the ethical conduct of research[11]. In particular, it adhered to requirements to respect the autonomy, rights and welfare of participants and minimise risk to them. The personal data that was collected for the study was minimal and cannot be used to personally identify any individual. Moreover, all the files used in the study were stored on an online repository provided by Anglia Ruskin. The only exceptions to this are the questionnaire and the data that it was used to collect, which were stored on the Joint Information Systems Committee's (JISC) Online Surveys platform[12].

8.5. Procedure

The developers were given one week to read the participant information sheet and return the signed consent form, which indicated their agreement to take part in the study. In addition, they were given two weeks to complete and submit the questionnaire. Out of the 50 developers who were solicited to participate in the study, 13 completed and submitted a questionnaire for a 26% response rate.

All of the data that had been collected during the study was collated on the JISC Online Surveys platform and checked for its integrity (e.g., missing values or that the questionnaires had been completed as expected). Afterwards, the data was exported to the Statistical Package for the Social Sciences (SPSS) for analysis.

---

[11] The research was approved by Anglia Ruskin's departmental (Computing and Technology) and faculty-level (Science and Technology) Research Ethics Panels for stage 1 and 2 types of research (i.e., covering up to risk category 3).

[12] The JISC Online Surveys platform is Anglia Ruskin's recommended platform for hosting online surveys.

8.6. Participant Experience Information

As mentioned earlier, some data was collected about the experiences of the participants of the study in designing and developing content for VWs. In particular, data was collected about the number of years of experience that the participants of the study have in designing and developing content for VWs. In addition, data was collected about the VW platforms that they are experienced in using to design and develop content for VWs. Table 8.4 shows a summary of the participant experience information.

Table 8.4. A summary of the participant experience information from the study

Experience characteristics		Frequency	Percentage
Experience (years)	Less than 1 year	0	0.0
	1 year or more, but less than 5 years	0	0.0
	5 years or more, but less than 10 years	5	38.5
	10 years or more, but less than 15 years	5	38.5
	15 years or more	3	23.1
Experience (VW platforms)	Active Worlds	6	46.2
	Second Life	13	100
	Open Simulator	7	53.8
	High Fidelity	5	38.5
	Meshmoon	4	30.8
	Sansar	13	100
	Other VW platform(s)	7	53.8

All the participants of the study have at least 5 years of experience designing and developing content for VWs. Out of all of them, only 3 have been designing and developing content for VWs for 15 years or more. Given that the participants of the study were recruited through Second Life and Sansar, the results are as expected, since neither platform is older than 15 years. However, it is common for developers to have experience in designing and developing content for a number of different VW platforms. As such, given that Active Worlds is 23 years old, it is also not surprising for some of the participants of the study to have reported having at least 15 years (or more) of experience in designing and developing content for VWs. The technical groups through which the participants were solicited are expert groups. Therefore, another result that was not surprising is that none of the participants reported having less than 5 years of experience in designing and developing content for VWs.

All of the participants of the study have experience in designing and developing content for Second Life and Sansar. Once again, this result is expected given that these were the same platforms through which recruitment of the developers was organised. Furthermore, given its popularity, it would be expected that a high number of the participants have experience in designing and developing content for Second Life. Sansar is one of the newest VW platforms. However, according to the results, all of the participants of the study have experience in designing and developing content for it. This is in contrast to the other older VW platforms, for which they generally have less experience in designing and developing content. An explanation for this is that, for many developers, Sansar represents a natural progression from Second Life. Sansar is new. However, it has already become popular for designing and developing content because it is a next-generation VW platform. Sansar's popularity among developers may also be helped by the fact that it was developed by Linden Lab, the makers of Second Life. Open Simulator follows after Second Life and Sansar as the platform for which most of the participants in the study have experience in designing and developing content. It is true that this platform has been the open source alternative to Second Life for many years now (White, 2008). The VW platform for which the participants of the study have the least experience in designing and developing content is Meshmoon. Like Sansar, Meshmoon is new and has been slowly trying to establish itself in the domain of VWs.

8.7. Reliability Measures

Prior to analysing the data that had been collected during the study, it was necessary to determine the reliability of the questionnaire that had been used to collect it. In particular, Cronbach's alpha[13] was used to measure the closeness of the statements in each of the

[13] Cronbach's alpha is a statistical function for measuring the internal consistency of an instrument (Heo, et al., 2015).

sets of statements that are related to fitness and utility. Table 8.5 shows a summary of the reliability of the questionnaire.

Table 8.5. Reliability of the questionnaire (Cronbach's alpha)

Criteria	Characteristics	Alpha
Fitness	7	0.823
Utility	6	0.934
Fitness-Utility	13	0.926

As shown in Table 8.5, the alpha for the fitness criteria is 0.823, which suggests that its items (i.e., the group of statements that are related to fitness on the questionnaire) have a high level of internal consistency[14]. Similarly, the internal consistency for the utility criteria is high at 0.934. Based on the consistency scores for fitness and utility, it was expected for fitness-utility to have a relatively high score, which is the case at 0.926.

8.8. Results

In SPSS, the data from the study was organised into variables that represent the fitness-utility statements on the survey questionnaire. Each variable contained all the answers (i.e., ratings) that were given by participants of the study for the fitness-utility statement that relates to it (see Appendix F). Since the answers are ordinal type data, they were coded as follows: 1 - Strongly Disagree, 2 - Disagree, 3 - Neither Agree nor Disagree, 4 - Agree and 5 - Strongly Agree.

The first step in analysing the data in SPSS was to determine the frequencies of the responses to each of the fitness-utility statements[15], the result of which are as follows:

The VWADM is made up of components that may be considered independent (or nearly independent) from each other, but work together during the design process.

[14] An alpha of 0.70 is generally considered acceptable (Tavakol & Dennick, 2011).
[15] This approach was used because it helps to determine the most frequent responses for each of the fitness-utility statements.

None of the participants of the study disagreed with the decomposability of the VWADM. Of the 13 participants, 8 of them agreed with its decomposability and 5 strongly agreed with it. Fig. 8.2 shows a summary of these results.

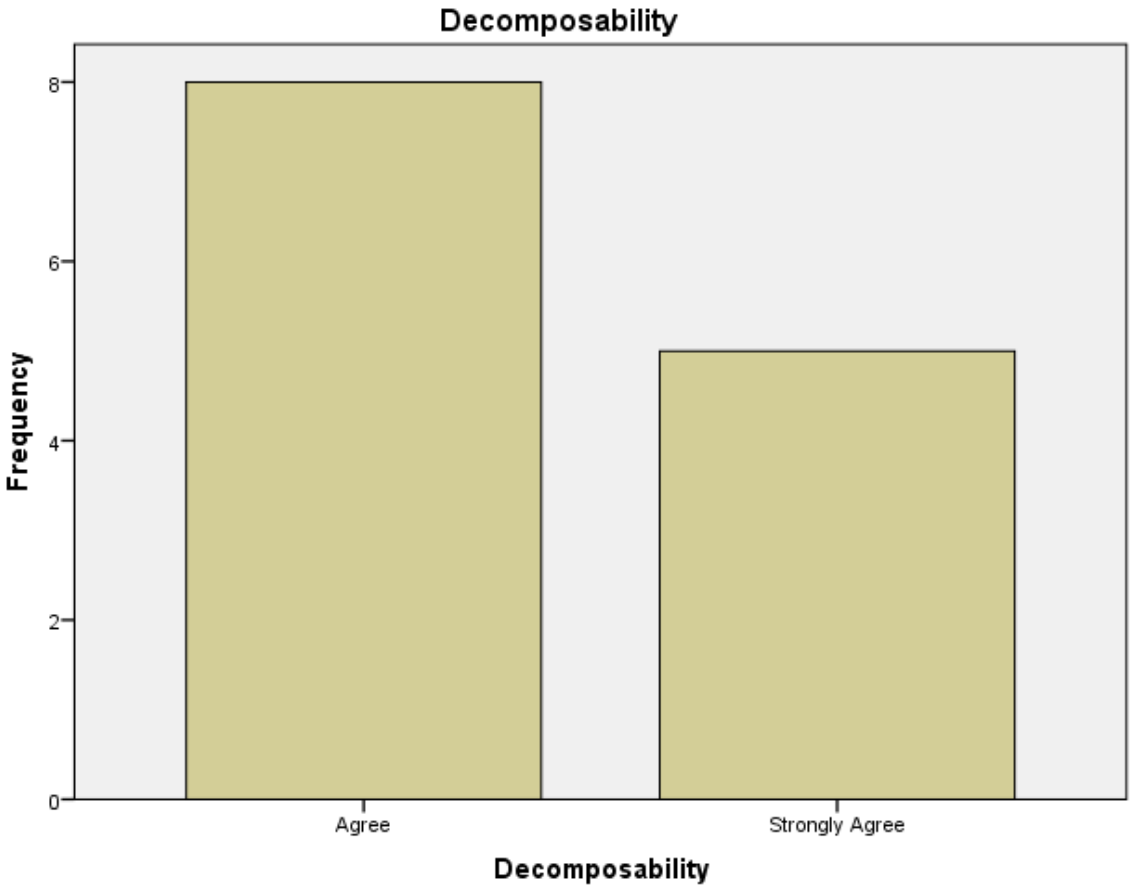


Figure 8.2. A summary of the opinions of participants of the study about the VWADM's decomposability

The VWADM is AT LEAST ONE of the following: (1) adaptable, (2) extensible or (3) customisable.

None of the participants of the study disagreed with the malleability of the VWADM. Of the 13 participants, 9 of them agreed with its malleability and 4 strongly agreed with it. Fig. 8.3 shows a summary of these results.

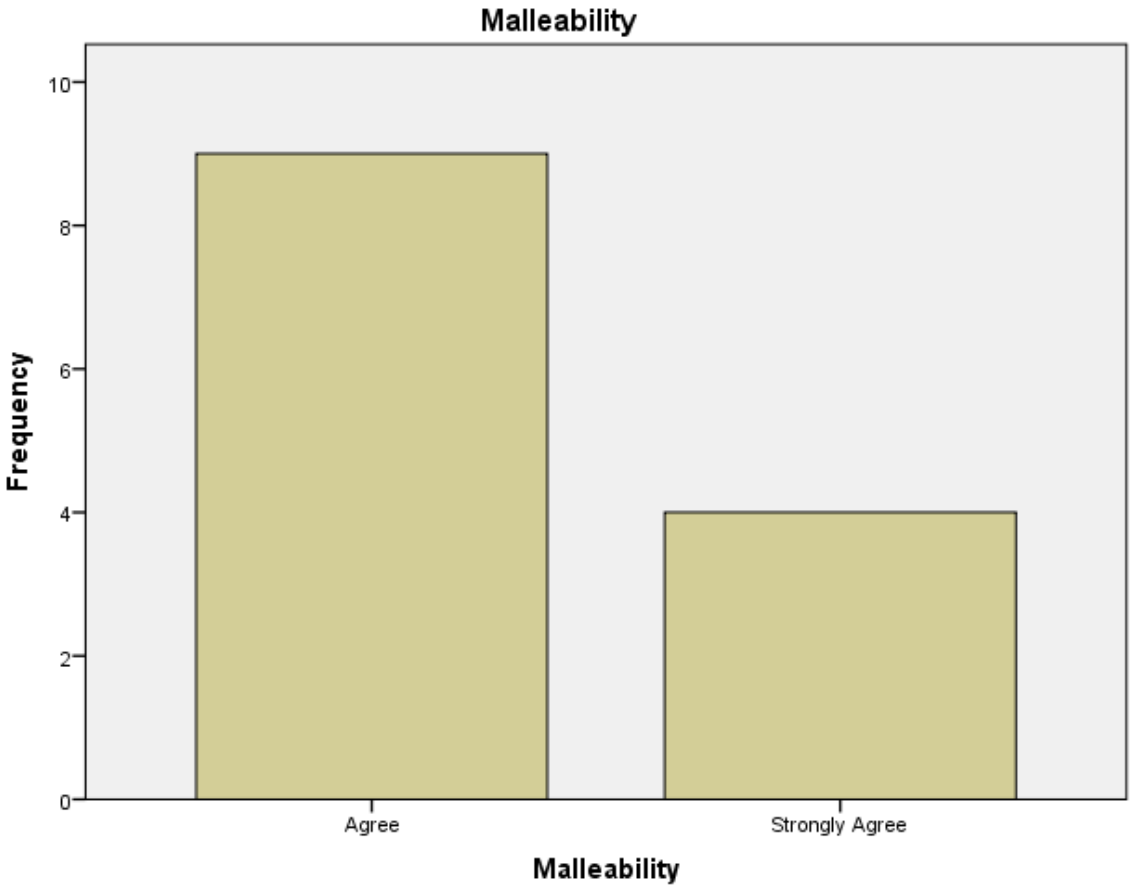


Figure 8.3. A summary of the opinions of participants of the study about the VWADM's malleability

The VWADM is AT LEAST ONE of the following: (1) open to inspection (2) open to modification or (3) open to reuse.

None of the participants of the study disagreed with the openness of the VWADM. Of the 13 participants, 9 of them agreed with its openness and 4 strongly agreed with it. Fig. 8.4 shows a summary of these results.

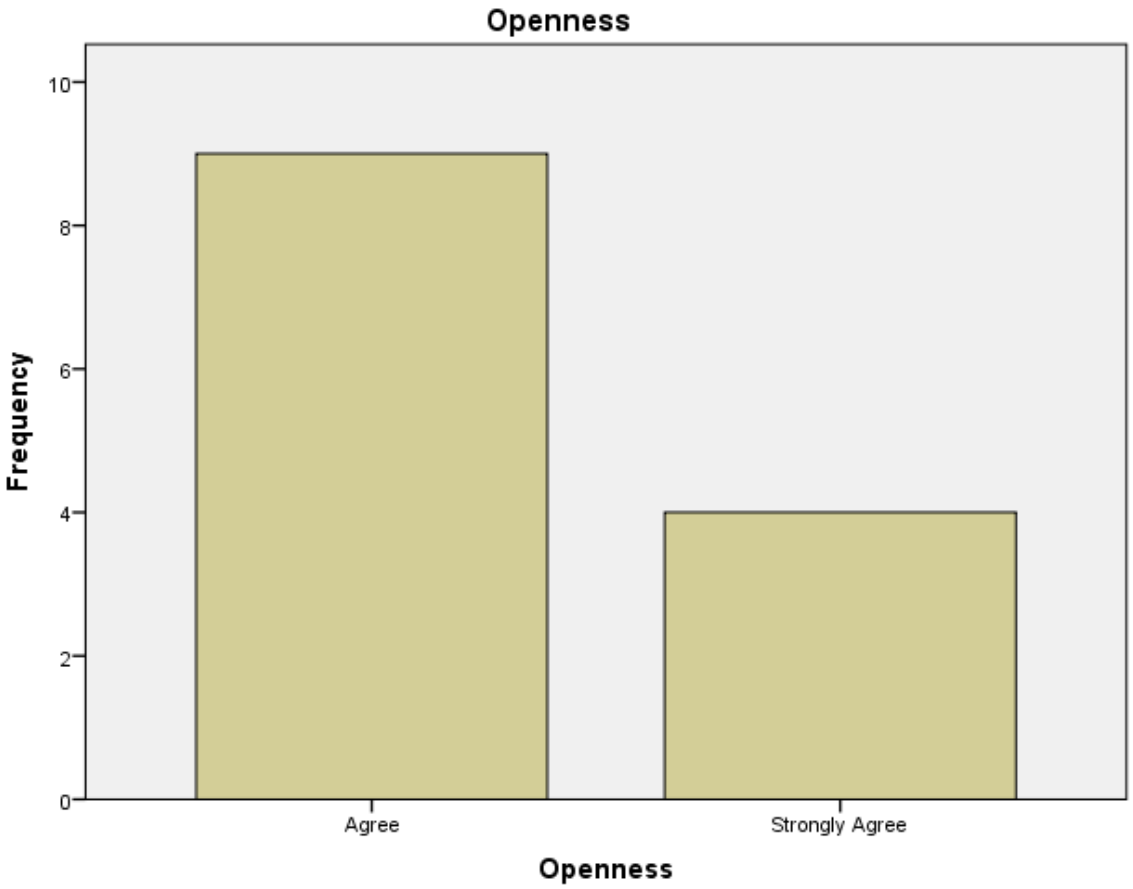


Figure 8.4. A summary of the opinions of participants of the study about the VWADM's openness

The VWADM originates from (or is inspired by) a field or environment in which design (i.e., software or non-software design) is routinely practiced.

Out of the 13 participants of the study, 7 of them agreed with the premise that the VWADM is embedded in a system where design is routinely practiced and 4 strongly agreed with it. Regarding the same premise, 1 of the participants disagreed with it and 1 was neutral about it. Fig. 8.5 shows a summary of these results.

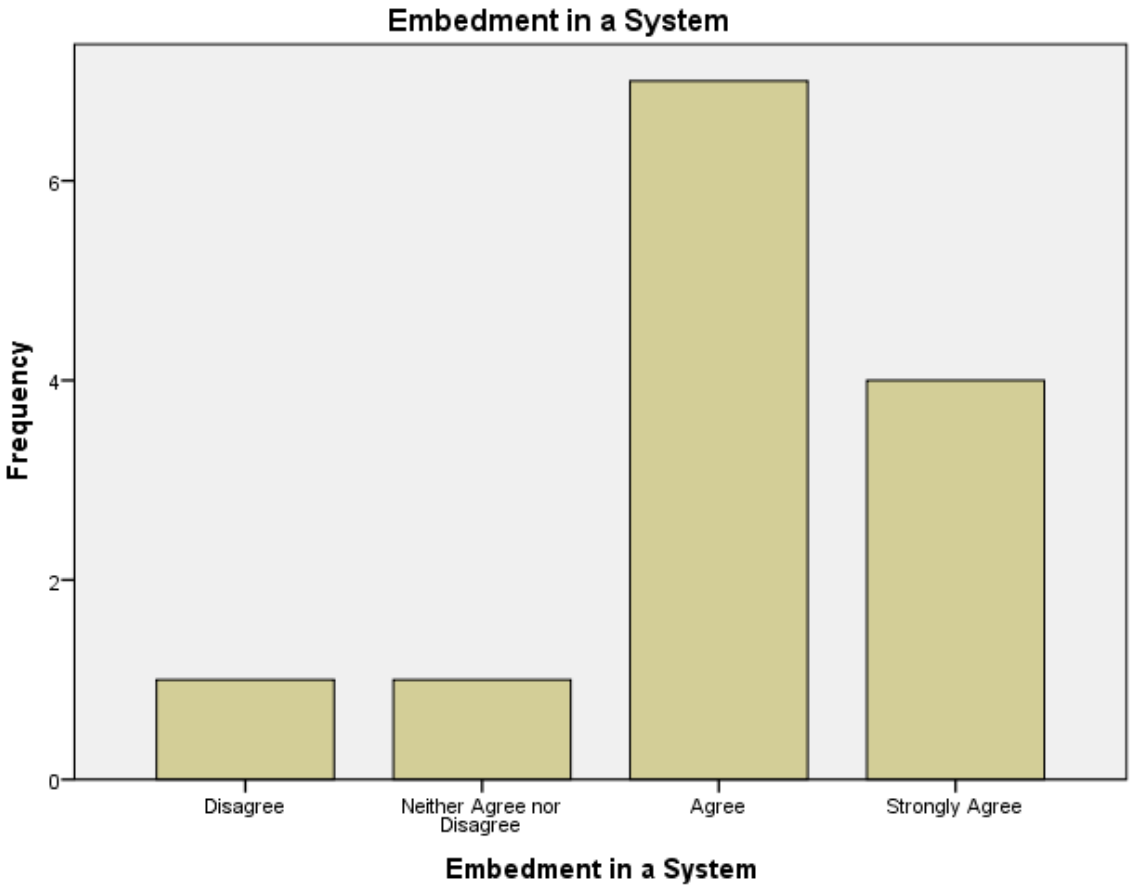


Figure 8.5. A summary of the opinions of participants of the study about the VWADM's embedment in a design system

The VWADM originates from (or is inspired by) a field that is different from software engineering.

Out of the 13 participants of the study, 8 of them agreed with the novelty of the VWADM and 3 strongly agreed with it. Regarding the same characteristic, 1 of the participants disagreed with it and 1 was neutral about it. Fig. 8.6 shows a summary of these results.

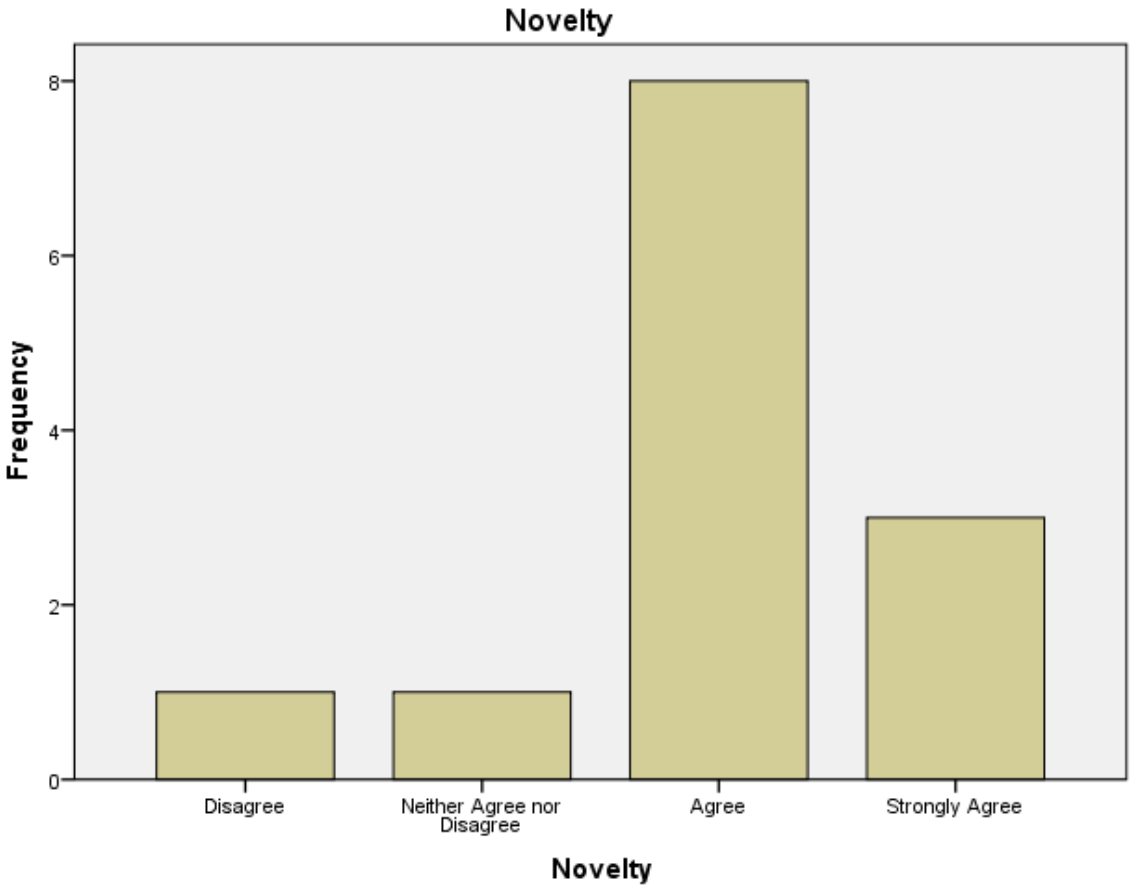


Figure 8.6. A summary of the opinions of participants of the study about the VWADM's novelty

The VWADM is interesting.

Out of the 13 participants of the study, 7 of them agreed with the novelty of the VWADM and 5 strongly agreed with it. Regarding the same characteristic, 1 of the participants was neutral about it. Fig. 8.7 shows a summary of these results.

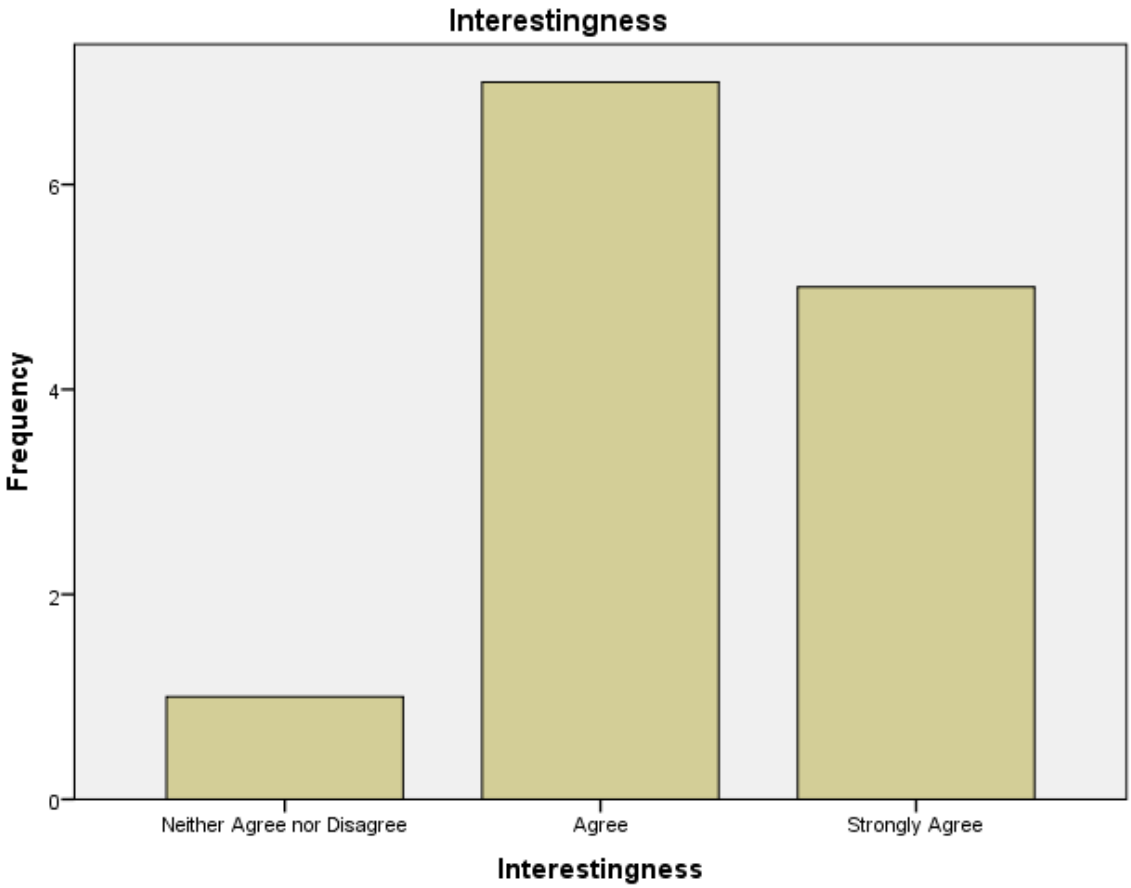


Figure 8.7. A summary of the opinions of participants of the study about the VWADM's interestingness

AT LEAST ONE of the following can be said about the VWADM: (1) it is compact, (2) it is simple, (2) its use is transparent, (3) its behaviour is transparent or (5) it has clarity of representation.

Out of the 13 participants of the study, 10 of them agreed with the elegance of the VWADM and 2 strongly agreed with it. Regarding the same characteristic, 1 of the participants disagreed about it. Fig. 8.8 shows a summary of these results.

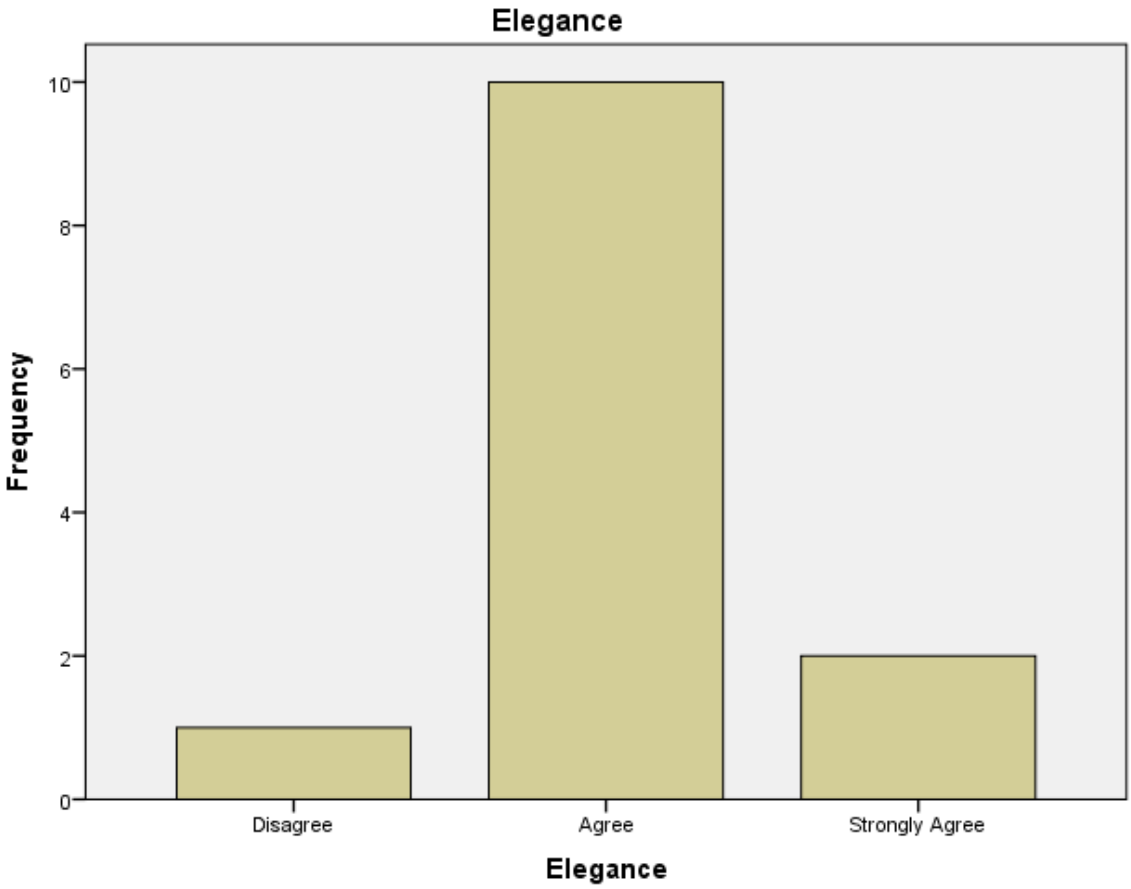


Figure 8.8. A summary of the opinions of participants of the study about the VWADM's elegance

Using the VWADM would enable developers to create place models of these applications more quickly during design.

Out of the 13 participants of the study, 3 of them agreed that the use of the VWADM would enable developers to create place models more quickly during the design process and 4 strongly agreed with this premise. Regarding the same premise, 6 of the participants were neutral about it. Fig. 8.9 shows a summary of these results.

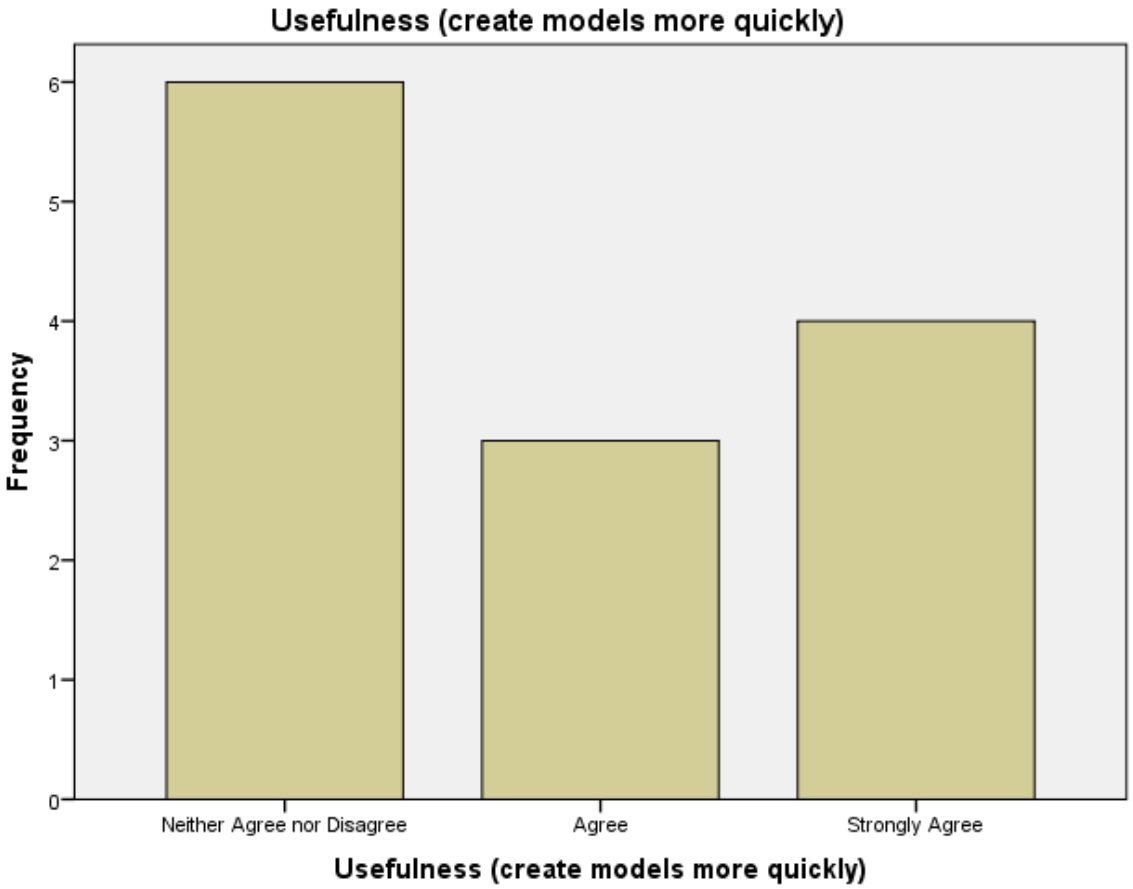


Figure 8.9. A summary of the opinions of the participants of the study about the VWADM's perceived usefulness in enabling developers to create place models more quickly during the design process

Using the VWADM would improve the performance of developers in creating place models of these applications during design.

Out of the 13 participants of the study, 2 of them agreed that the use of the VWADM would improve the performance of developers in creating place models of VW applications more quickly during the design process and 2 strongly agreed with this premise. Regarding the same premise, 3 of the participants were neutral about it and 6 disagreed with it. Fig. 8.10 shows a summary of these results.

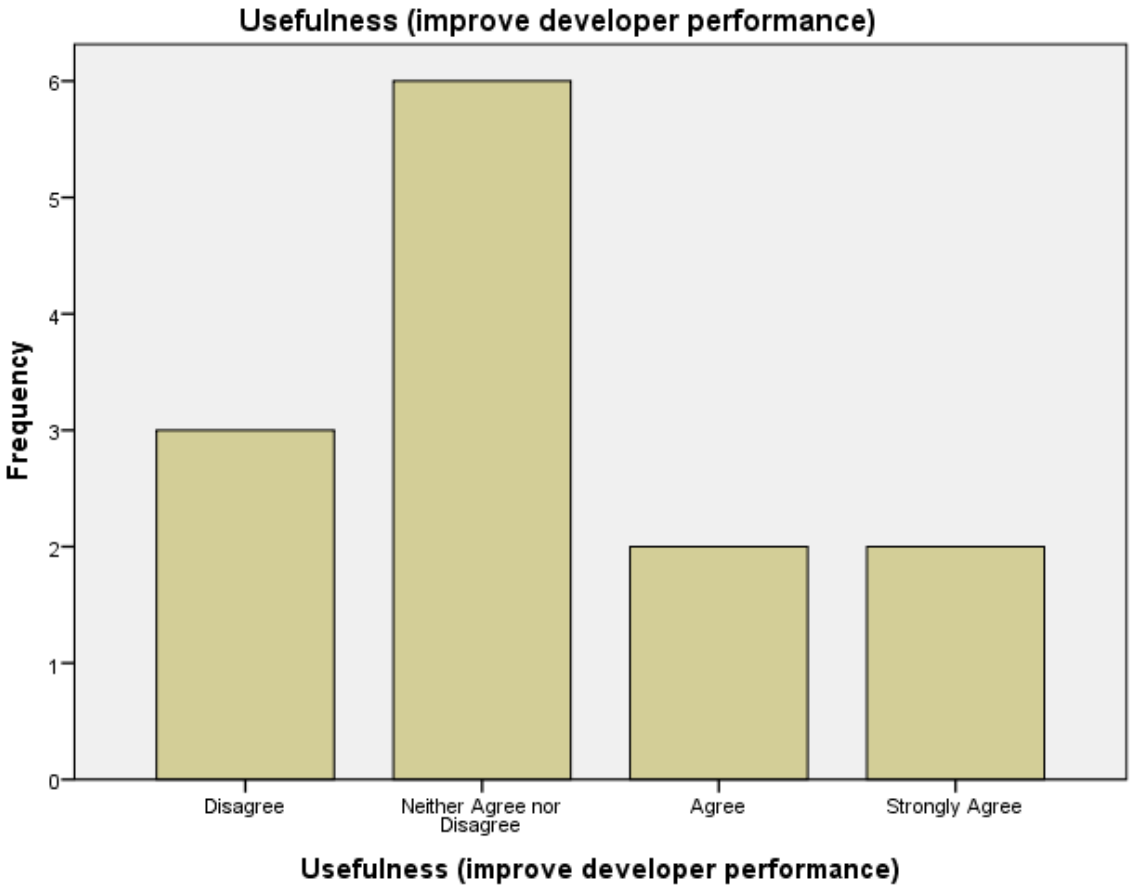


Figure 8.10. A summary of the opinions of the participants of the study about the VWADM's perceived usefulness for improving the performance of developers during the design process

Using the VWADM would increase the productivity of developers when creating place models of these applications during design.

Out of the 13 participants of the study, 1 of them agreed that the use of the VWADM would increase the productivity of developers during the design process and 3 strongly agreed with this premise. Regarding the same premise, 2 of the participants disagreed with it, 1 strongly disagreed with it and 6 were neutral about it. Fig. 8.11 shows a summary of these results.

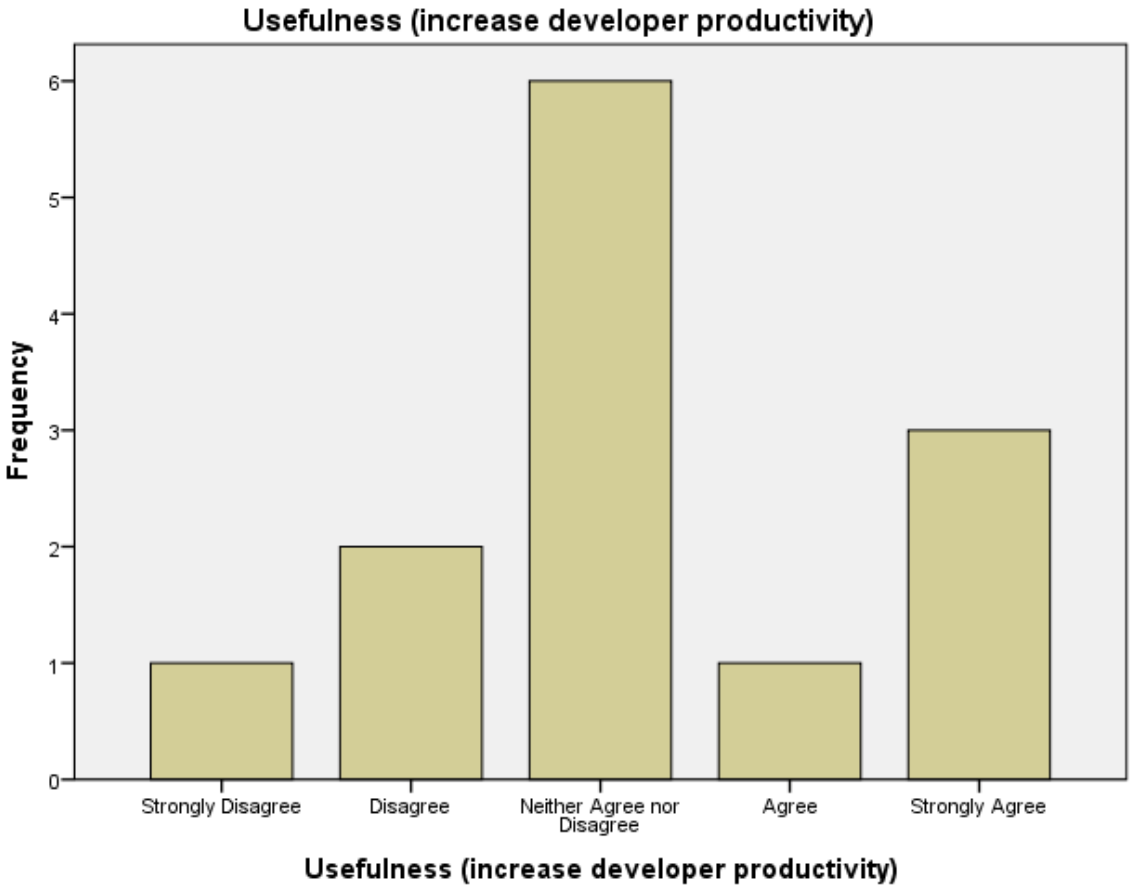


Figure 8.11. A summary of the opinions of the participants of the study about the VWADM's perceived usefulness for increasing the productivity of developers during the design process

Using the VWADM would enhance the effectiveness of developers in creating place models of these applications during design.

Out of the 13 participants of the study, 8 of them agreed that the use of the VWADM would increase the productivity of developers during the design process and 1 strongly agreed with this premise. Regarding the same premise, only 1 of the participants disagreed with it and 3 were neutral about it. Fig. 8.12 shows a summary of these results.

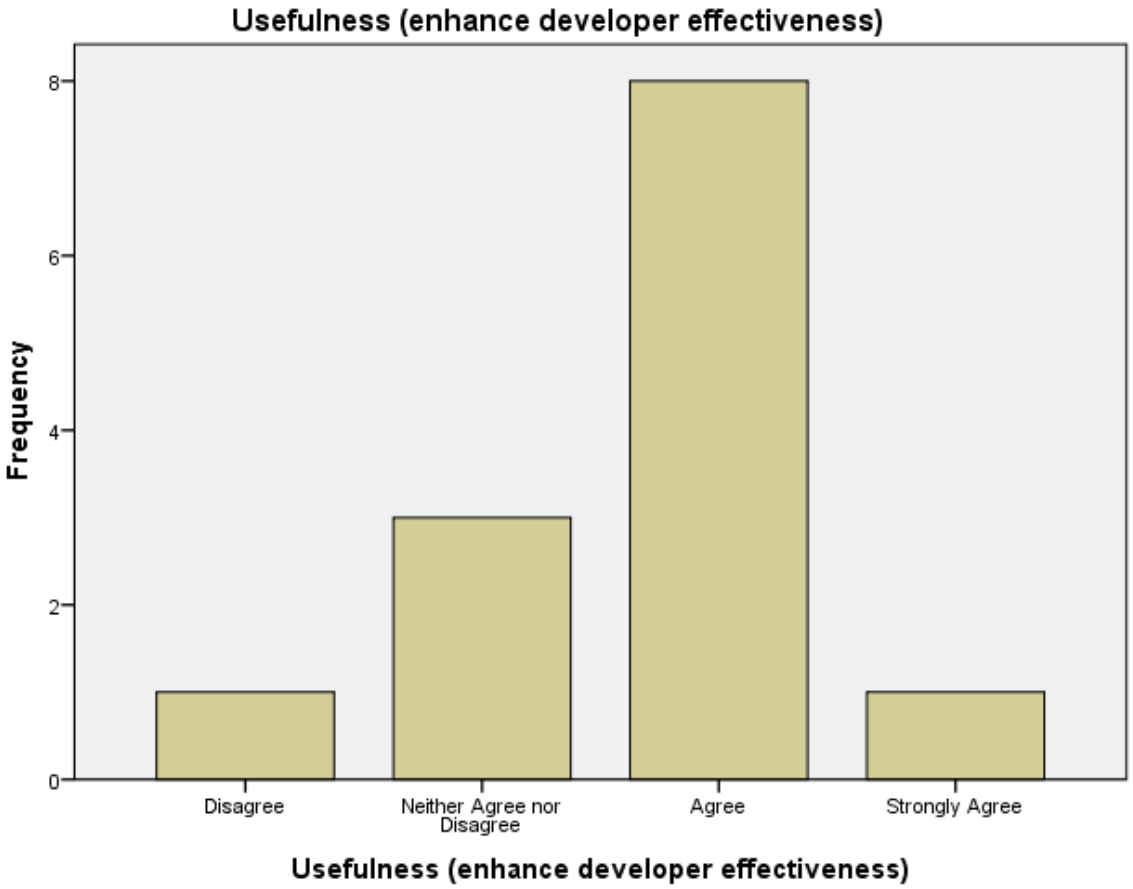


Figure 8.12. A summary of the opinions of the participants of the study about the VWADM's perceived usefulness for enhancing the effectiveness of developers during the design process

Using the VWADM would make it easier for developers to create place models of these applications during design.

Out of the 13 participants of the study, 3 of them agreed that the use of the VWADM would increase the productivity of developers during the design process and 6 strongly agreed with this premise. Regarding the same premise, 4 of the participants were neutral about it. Fig. 8.13 shows a summary of these results.

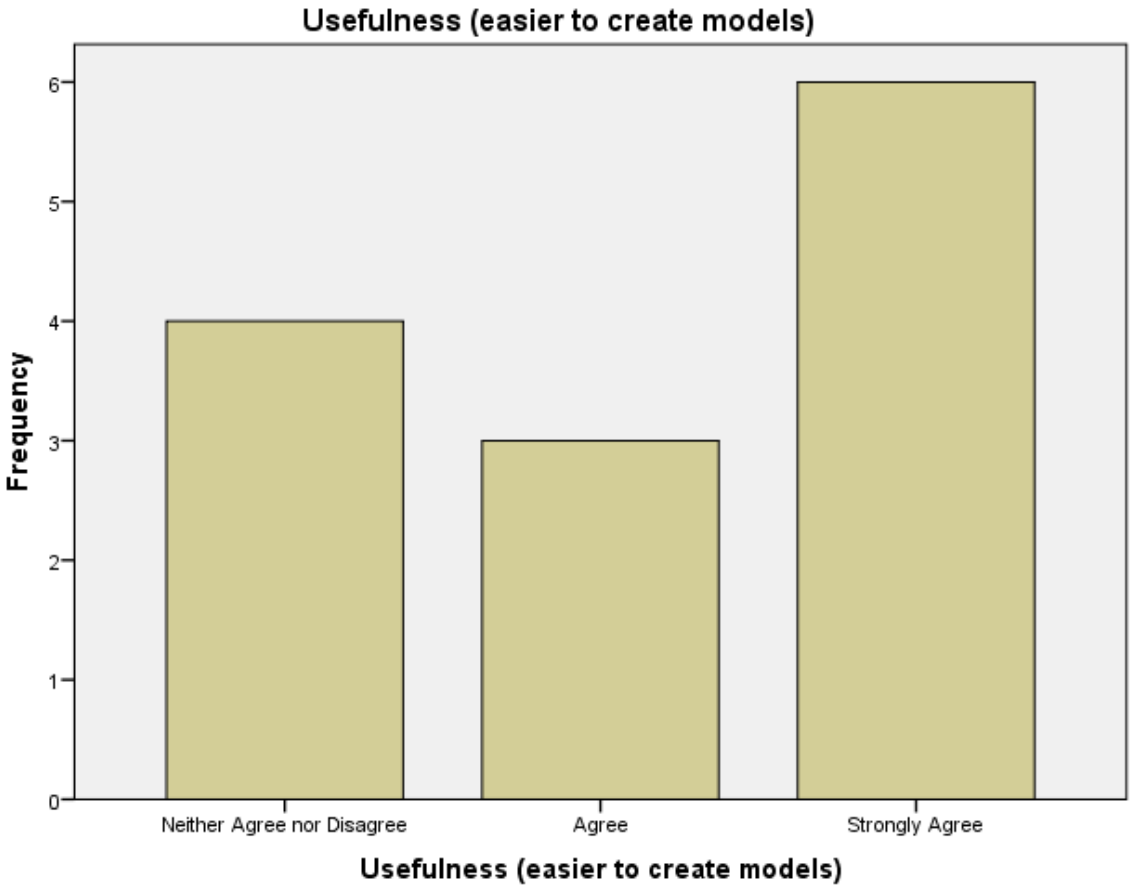


Figure 8.13. A summary of the opinions of the participants of the study about the VWADM's perceived usefulness for facilitating easier creation of place models

Developers would find the VWADM useful for creating place models of VW applications during design.

Out of the 13 participants of the study, 5 of them agreed that the VWADM would be useful (to developers) for creating place models of VW applications during the design process and 4 strongly agreed with this premise. Regarding the same premise, 4 were neutral about it. Fig. 8.14 shows a summary of these results.

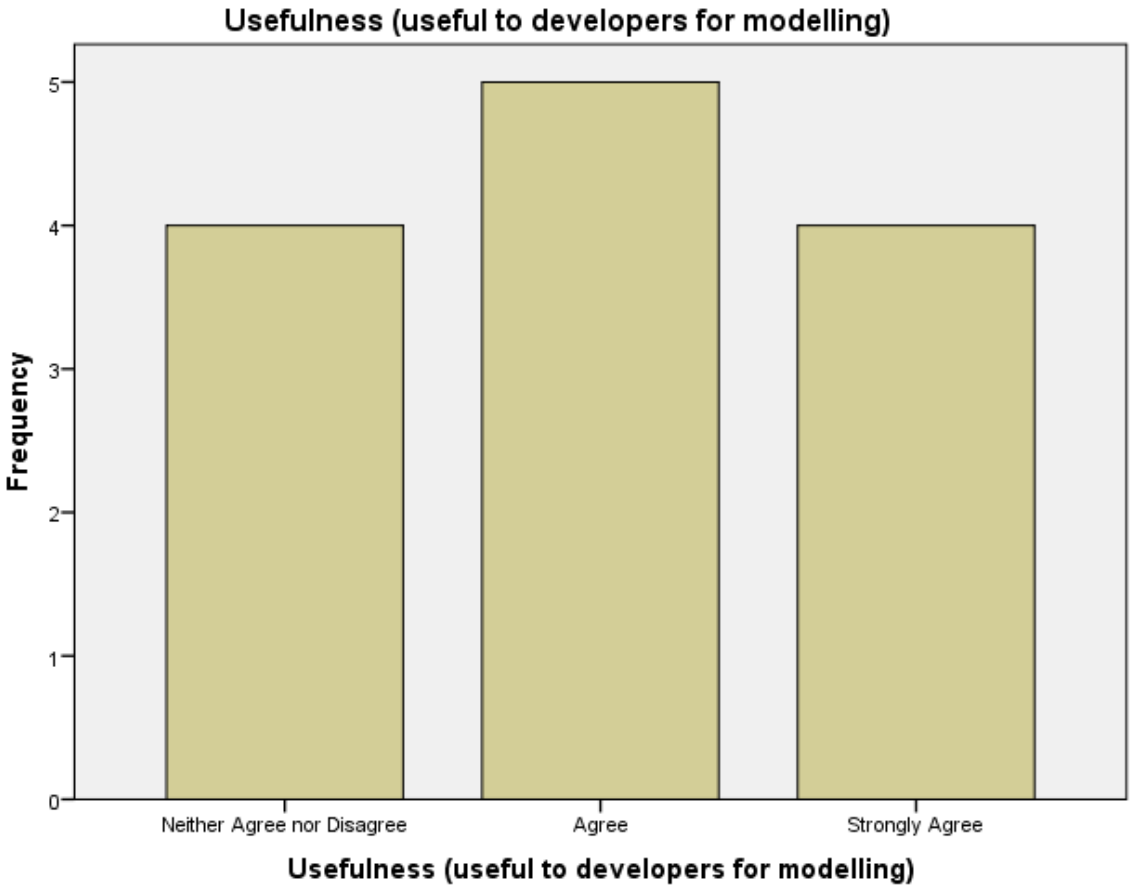


Figure 8.14. A summary of the opinions of the participants of the study about the VWADM's perceived usefulness to developers

The next step in analysing the data was to use the frequencies of the responses to the fitness-utility statements to determine a weighting for the fitness and utility criteria. In order to do so, two composite variables were created in SPSS to each hold a set of fitness and utility-related responses (resulting in one variable that measures fitness and another that measures utility). The median was taken for each variable[16], the results of which are shown in Table 8.6 as follows:

Table 8.6. A summary of the results for fitness and utility

Criteria	Median	*n* Valid	Missing
Fitness	4.00	13	0
Utility	3.50	13	0

[16] This approach was used to determine the central tendency of the data. The median is used to statistically adjust for extreme answers (i.e., Strongly Disagree and Strongly Agree) and neutral answers (i.e., Neither Agree nor Disagree). By using the median, it becomes possible to deduce weightings for the fitness and utility of the VWADM.

The final step in analysing the data was to use the weightings of the fitness and utility criteria to determine a fitness-utility score. In order to do so, a composite variable was created in SPSS using the variables for fitness and utility. A one sample T test was performed on the fitness-utility variable[17], the result of which is shown in Table 8.7 as follows:

Table 8.7. A summary of the results for Fitness-Utility

Criteria	Mean	Std. Deviation	Std. Error Mean
Fitness-Utility	4.00	0.595	0.165

8.9. Discussion

The user study was conducted in order to evaluate the features and capabilities of the VWADM with respect to their significance for practical application in designing VW applications. The results suggest that the VWADM can provide a significant contribution to automating the process of VW design from the point of view of place-oriented design.

The developers who participated in the study agreed on the VWADM's fitness (answer median = 4.00 out of the range of 1.00 to 5.00) for creating place models of VW applications during the design process and somewhat agreed on its utility (answer median = 3.50 out of the range of 1.00 to 5.00) for the same purpose. The VWADM's low utility score was positively impacted by its high fitness score as indicated by the result of the t test (i.e., fitness-utility mean = 4.00 out of the range of 1.00 to 5.00)[18]. These results suggest that the features and capabilities of the VWADM are significant for practical application in designing VW applications.

[17] By using this approach, comparisons can be made between the mean for fitness-utility in this study and another known or hypothesised value of mean in similar studies.
[18] The influence of fitness on utility is also supported in the literature that pertains to fitness-utility (Sterelny, 2012)

The statistical significance of the sample size of the participants of the study is ensured by a number of aspects. Firstly, similar studies such as those conducted by (Ruddle & Peruch, 2004; Vanacken, et al., 2007) used sample sizes of a dozen or less. In contrast, this study used a sample size that is slightly larger (i.e., n = 13). Secondly, the population of developers who are involved in content creation for VWs is a closed group, which makes requirements for the sample size much smaller than other general studies in SE. Thirdly, the data that was collected suggests that there is 90% confidence that the population mean (μ) for fitness-utility will fall between 3.06 and 4.94 (out of a range of 1.00 to 5.00)[19]. Finally, based on the high levels of experience of the participants of the study, it is possible that members of the population who have the same level of experience will share similar opinions about the VWADM's fitness and utility for creating place models of VW applications during the design process.

8.10. Summary

This chapter discussed a user study that was conducted to evaluate the features and capabilities of the VWADM with respect to their significance for practical application in designing VW applications. The chapter began by providing an overview of the fitness-utility model, which was used to design the study and formulate the criteria for evaluation. Next, it provided an overview of the methods and instruments that were chosen for capturing the opinions of developers during the study. This was followed by a statement on the preparatory activities that were performed to recruit developers to participate in the study. The statement on the preparatory activities of the study was followed by discussions of some ethical considerations and the procedure that was used to conduct the study. These two sections were followed by information from the study, which includes participant

[19] The confidence interval was calculated using the sample mean (i.e., fitness-utility mean = 4.00), sample size (i.e., n = 13) and standard deviation (i.e., s = 2.05) and the confidence level of 90%.

experience information, a report on the reliability of the questionnaire used in the study and the results of the study. Finally, the results of the study were discussed.

9. Conclusion

This chapter summarises the thesis and evaluates the work in context of the research hypothesis. Some of the limitations of the research are reviewed, as are some of the benefits of the VWADM for developers. Thoughts on future related work are also presented.

9.1. Summary of the Thesis

This thesis presented the VWADM, which is a new method that was developed for designing VW applications. The VWADM combines a GDG framework and a GDA model to form the template of a mechanism that can be used for developing GDGs and GDAs. GDGs can be used as tools for capturing and storing spatial data about VW applications and using it to generate place-oriented conceptual models of these applications. In a complementary fashion, GDAs wrap around GDGs, encapsulating them and providing the means to dynamically generate physical models of VW applications.

The research developed the VWADM using techniques from architecture and applied it as a novel solution in SE to the problem of designing VW applications. The VWADM provides support for the place-oriented viewpoint and can be used by developers for modelling VW applications during design. As such, the research makes the following contributions to knowledge:

- It establishes a set of modelling concepts that use grammar and rules as the basis for capturing the semantics of the conceptual models of VW applications.
- It establishes a set of visual notations, which include basic shapes and symbols that can be used to present the conceptual models of VW applications to stakeholders for them to manually review.

- It establishes an intelligent component that is used for automatically generating physical models of VW applications.
- It establishes a stepwise process, which involves layout design, object design, navigation design and interaction design and acts as a guide for the process of creating the conceptual and physical models of VW applications.

9.2. Review of the Research Hypothesis

The research shows that the VWADM has the functionality for creating place-oriented conceptual and physical models of VW applications and is of use to developers for designing these types of applications. This supports hypothesis H1, which states that **a method can be developed, which has the functionality for creating place-oriented conceptual and physical models of VW applications and be of use to developers for designing these types of applications**. In particular, the case studies in Chapter 7 show that the VWADM has the functionality for creating place-oriented conceptual and physical models of VW applications. This supports the sub-hypothesis H1a, which states that **a method can be developed, which has the functionality for creating place-oriented conceptual and physical models of VW applications**. Furthermore, the results of the evaluation (Chapter 8) show that the VWADM is useful to developers and therefore supports the sub-hypothesis H1b, which states that **a method can be developed, which is useful to developers for creating place-oriented conceptual and physical models of VW applications during design**.

9.3. Limitations of the Research

The VWADM was developed to provide support for the place-oriented viewpoint that pertains to VW applications. However, there are many other viewpoints that pertain to these applications (e.g., process-oriented viewpoint, social network-oriented viewpoint, API-oriented viewpoint or UI-oriented viewpoint) and there is the potential for new ones to emerge. Therefore, more methods (and certainly a diverse range of them) are required for designing VW applications in terms of other viewpoints. To this end, the VWADM is not

intended to compete with, replace or improve on the performance of existing methods in SE that are used for designing VW applications. Instead, the intention is for it to complement them (as well as any other methods that may be developed in the future) and to contribute to widening the range of viewpoints that are supported in the design process for VW applications.

With regards to the study that was conducted as part of the evaluation of the VWADM, it could be improved in a few ways. Firstly, the opinions of developers about the fitness-utility characteristics would be more revealing if the questionnaire is comprised of open-ended questions (and/or complemented by interviews). Secondly, it would be interesting to evaluate how well the VWADM (or an implementation that is based on its ideas) works in VWs other than those that were used in the context of the research that this thesis is based on. Finally, although a likely weaker form of evaluation, the features and capabilities of existing methods in SE could be used as a benchmark for a comparative evaluation of the VWADM with existing methods in SE to determine its suitability to enter the design space for VW applications.

9.4. Benefits of the VWADM for Developers

The major benefit of the VWADM for developers is that it will enable them to automate the process of designing VW applications. Based on the results of the evaluation, this will:

- enable developers to create place models of VW applications more quickly
- improve the performance of developers in creating place models of VW applications
- increase the productivity of developers when creating place models of VW applications
- enhance the effectiveness of developers in creating place models of VW applications
- make it easier for developers to create place models of VW applications
- provide a useful framework (that includes methods and tools) to developers to enable them to create place models of VW applications

9.5. Future Work

One of the potential future directions of the research is to work with developers to incorporate the features of the VWADM into their workflow. Examples of VW applications in which the VWADM could be used include learning environments that can dynamically adapt to learners and the activities they perform in such environments (Ewais & Troyer, 2014; Scott, et al., 2017) or a type of virtual prototyping in which agents are responsible for brainstorming about the various potential designs (Fougeresa & Ostrosi, 2018) and developers decide on the final choice of the design.

Current research shows that VR is a promising technology that can be used by clinicians to assess, understand and treat mental disorders such as anxiety, obsessive-compulsive disorder (OCD) and schizophrenia (Freeman, et al., 2017). It is also possible for VR to be used by therapists to provide next-generation therapy for people suffering from such mental disorders (Rus-Calafell, et al., 2015; Craig, et al., 2017). Therefore, another potential future direction of the research is to work with clinicians, therapists and other mental health practitioners to develop scenarios based on treatment (or therapy) regimes. Such scenarios would then become the basis for rules that can be used for generating VW applications for treating people with mental disorders. An example of a VW application in which the VWADM could be used for treating people with mental disorders is one that uses GDAs to monitor anxiety (via sensors), regulates exposure to a scenario (via a hypothesising mechanism) and (re)generates the appropriate VW application (via effectors) based on techniques used in cognitive behavioural therapy.

A third potential future direction of the research is to incorporate features of the VWADM into a bespoke VW and to conduct evaluations on the implication. This approach would serve as the basis for intelligent desktop VR systems. An example VW application that could be explored in this context is intelligent scalable land (Hendrikx, et al., 2013; Steinberger, et al., 2014; Diaconu & Keller, 2016) that can accommodate real-time changes in the user

population in VWs and uses generative design techniques for managing (i.e., allocating and reclaiming) its limited resources.

9.6. Outlook

The development of intelligent desktop VR systems is an idea that has the potential to contribute immensely to the democratisation of the design process in VWs. That said, a future is envisioned in which the design of content in VWs involves many different groups of users with varying technical abilities, especially those who belong to limited access groups. At such a time, users will only be concerned with passing on their design requirements to agents who will be tasked with generating content based on those requirements.

9.7. Final Remarks

The AW agent package is being released on Anglia Ruskin's research repository, Anglia Ruskin Research Online (ARRO), under the MIT licencing terms. Use of the files that comprise it is only possible in AW in conjunction with the Jess rule engine and the Java programming language. Nevertheless, it is also possible to use the source code and guides to implement similar mechanisms in other desktop VR systems. For example, LSL has built-in functions and events that support the implementation of sensors (e.g., llSensor(), llSensorRepeat() and sensor()), which can be used in conjunction with state-based rules and functions to programmatically spawn objects (e.g., llRezObject()) in Second Life.

References

ActiveWorlds Inc., 2014. *SDK.* [Online] Available at: http://wiki.activeworlds.com/index.php?title=SDK [Accessed 09 October 2017].

AEM, 2016. *How Artificial Intelligence Could Revolutionize Construction.* [Online] Available at: https://www.aem.org/news/october-2016/how-artificial-intelligence-could-revolutionize-construction/ [Accessed 14 May 2017].

Archer, L. B., 1984. Systematic Method for Designers. In: N. Cross, ed. *Developments in Design Methodology.* London: John Wiley, pp. 57-82.

Austin, J. L., 1962. *How to do Things with Words.* 1st ed. London: Oxford University Press.

Bainbridge, W. S., 2014. Gaming and Virtual Worlds. In: *Encyclopedia of Social Network Analysis and Mining.* New York, NY: Springer, pp. 599-609.

Barfield, W., ed., 2016. *Fundamentals of Wearable Computers and Augumented Reality.* 2nd ed. Boca Raton, FL: CRC Press.

Barrett, T., 2011. *Making Art: Form & Meaning.* 1 ed. New York City, NY: McGraw-Hill.

Bell, J. et al., 1994. *Software design for reliability and reuse: a proof-of-concept demonstration.* Baltimore, MD, ACM, pp. 396-404.

Billinghurst, M., Grasset, R. & Looser, J., 2005. Designing Augmented Reality Interfaces. *Computer Graphics,* 39(1), pp. 17-22.

Boehm, B., 2000. *Spiral Development: Experience, Principles, and Refinements,* Pittsburgh, PA: Carnegie Mellon.

Boehm, B. W., 1988. A Spiral Model of Software Development and Enhancement. *Computer,* pp. 61-72.

Bowen, S. J., 2009. *A Critical Artefact Methodology: Using Provocative Conceptual*

Designs to Foster Human-Centred Innovation, Sheffield: Sheffield Hallam University.

Bowman, D. A., Kruijff, E., LaViola Jr, J. J. & Poupyrev, I., 2001. An Introduction to 3-D User Interface Design. *Presence: Teleoperators and Virtual Environments,* 10(1), pp. 96-108.

Brown, A. L., 1992. Design Experiments: Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings. *The Journal of the Learning Sciences,* 2(2), pp. 141-178.

Budde, R., Kautz, K., Kuhlenkamp, K. & Zuellighoven, H., 1992. Prototyping. In: *An Approach to Evolutionary System Development.* Berlin: Springer, pp. 33-46.

Burdea, G., Richard, P. & Coiffet, P., 1996. Multimodal Virtual Reality: Input-Output Devices, System Integration, and Human Factors. *International Journal of Human-Computer Interaction,* 8(1), pp. 5-24.

Caldas, L. & Rocha, J., 2001. *A Generative Design System Applied to Siza's School of Architecure at Oporto.* Sydney, University of Sydney, pp. 253-264.

Carvalho, M. B. et al., 2015. An Activity Theory-Based Model for Serious Games Analysis and Conceptual Design. *Computers & Education,* Volume 87, pp. 166-181.

Cheng, T. & Yeh, M., 2007. *A Scenario Driven Scheme for Rapid Modeling of Interactive Virtual Environments.* Valparaiso, IFPR.

Chevaillier, P., Trinh, T. & Barange, M., 2012. Semantic Modeling of Virtual Environments using MASCARET. In: *Proceedings of the 5th Workshop on Software Engineering and Architectures for Realtime Interactive Systems.* Orange County, CA: IEEE, pp. 1-8.

Chien, S. & Flemming, U., 1997. Information Navigation in Generative Design Systems. In: Y. Liu, J. Tsou & J. Hou, eds. *Proceedings of The Second Conference on Computer Aided Architectural Design Research In Asia.* Hsinchi: Hu's Publisher Inc, pp. 355-365.

Churchill, E. F., Snowdon, D. N. & Munro, A. J., 2001. *Collaborative Virtual Environments: Digital Places and Spaces for Interaction.* London: Springer.

Churchman, C. W., 1971. *The Design of Inquiriing Systems: Basic Concepts of Systems and Organization.* New York, NY: Basic Books.

Cicognani, A., 1998. *A Linguistic Characterisation of Design in Text-Based Virtual Worlds,* Sydney: Department of Architectural and Design Science, Faculty of Architecture, University of Sydney.

Cicognani, A., 1998. On the Linguistic Nature of Cyberspace and Virtual Communities. *Virtual Reality,* 3(1), pp. 16-24.

Cicognani, A., 2000. Language and Design in Text-Based Virtual Worlds. *Journal of Network and Computer Applications,* Volume 23, pp. 247-274.

Cicognani, A. & Maher, M. L., 1997. *Design Speech Acts. "How to Do Things with Words" in Virtual Communities.* Munich, Kluwer Academic Publishers, pp. 707-717.

Clarke, E. M. & Wing, J. M., 1996. Formal Methods: State of the Art and Future Directions. *Computing Surveys,* 28(4), pp. 626-643.

Constantine, L. L. & Yourdon, E., 1979. *Structued Design.* Englewood Cliffs, NJ: Prentice-Hall.

Craig, T. K. J. et al., 2017. Avatar Therapy for Auditory Verbal Hallucinations in People with Psychosis: A Single-Blind, Randomised Controlled Trial. *The Lancet Psychiatry,* pp. 1-10.

Craven, J., 2017. *What is a Floor Plan?.* [Online] Available at: https://www.thoughtco.com/what-is-a-floor-plan-175918 [Accessed 03 June 2018].

Craven, J., 2018. *The Three Rules of Architecture: How to Win a Pritzker Architecture Prize.* [Online] Available at: https://www.thoughtco.com/the-rules-of-architecture-177224 [Accessed 03 June 2018].

Cross, N., 2001. Desingerly Ways of Knowing: Design Discipline Versus Design Science. *Design Issues,* 17(3), pp. 49-55.

Cruz-Neira, C., Sandin, D. J. & DeFanti, T. A., 1993. *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE.* Anaheim, CA, ACM, pp. 135-142.

Curtis, B., Krasner, H. & Iscoe, N., 1988. A Field Study of the Software Design Process for

Large Systems. *Communications of the ACM,* 31(11), pp. 1268-1287.

Davis, F. D., 1989. Perceived Usefulness, Perceived Ease Of Use, and User Acceptance of Information Technology. *MIS Quarterly,* 13(3), pp. 319-340.

Dawson, M., Davis, L. & Omar, M., 2017. Developing Learning Objects for Engineering and Science Fields: Using Technology to Test System Usability and Interface Design. *International Journal of Smart Technology and Learning.*

DeMarco, T., 1978. *Structured Analysis and System Specification.* New York, NY: Yourdon Press.

Deshmukh, N. P. et al., 2015. Five-Dimensional Ultrasound System for Tissue Visualization. *The International Journal for Computer Assisted Radiology and Surgery,* 10(12), pp. 1927-1939.

Devedzic, V., Debenham, J. & Popovic, D., 2000. Teaching Formal Languages by an Intelligent Tutoring System. *Educational Technology & Society,* 3(2), pp. 36-49.

Diaconu, R. & Keller, J., 2016. *Kiwano: Scaling Virtual Worlds.* Las Vegas, NV, IEEE.

Diesing, P., 2017. *Patterns of Discovery in the Social Sciences.* 2nd ed. New York, NY: Routledge.

Dobrica, L. & Niemela, E., 2002. A Survey on Software Acrhitecture Analysis Methods. *Transactions on Software Engineering,* 28(7), pp. 638-653.

Dodgson, M., Gann, D. M. & Phillips, N., 2013. Organizational Learning and the Technology of Foolishness: The Case of Virtual Worlds at IBM. *Organization Science,* 24(5), pp. 1358-1376.

Eekels, J. & Roozenburg, N. F. M., 1991. A Methodological Comparison of the Structures of Scientific Research and Engiineering Design: Their Similarlities and Differences. *Design Studies,* 12(4), pp. 197-203.

El-Far, I. K. & Whittaker, J. A., 2001. Model-based Software Testing. In: J. J. Marciniak, ed. *Encyclopedia on Software Engineering.* 2nd ed. New York, NY: Wiley, pp. 1-22.

El-Khalili, N., 2009. Architectural Framework for Immersive Web Based Virtual

Environments. *The International Arab Journal of Information Technology,* 6(4), pp. 365-370.

Esping-Andersen, G., 1990. The Three Political Economies of the Welfware State. *International Journal of Sociology,* 20(3), pp. 92-123.

Ewais, A. & Troyer, O. D., 2014. Authoring Adaptive 3D Virtual Learning Environments. *International Journal of Virtual and Personal Learning Environments,* 5(1), pp. 1-19.

Figueroa, P. et al., 2008. InTml: A Dataflow Oriented Development System for Virtual Reality Applications. *Presence,* 17(5), pp. 492-511.

Finkelstein, A., Kramer, J. & Goedicke, M., 1990. *Viewpoint Oriented Software Development.* Toulouse, CERN, pp. 1-21.

Finkelstein, L., Huang, J., Finkelstein, A. C. W. & Nuseibeh, B., 1992. Using Software Specification Methods for Measurement Instrument Systems. *Measurement Journal,* 10(2), pp. 79-86.

Fischer, G., 1994. Domain-Oriented Design Environments. *Automated Software Engineering,* 1(2), pp. 177-203.

Flemming, U., 1995. Software Environments to Support Earsly Phases in Building Design. *Journal of Architectural Engineering,* 1(4), pp. 147-152.

Forcada, N. et al., 2015. A BIM Model to Visualize Quality Risks in Construction Projects. In: A. Mahdavi, B. Martens & R. Scherer, eds. *eWork and eBusiness in Architecture, Engineering and Construction.* London: Taylor & Francis Group, pp. 69-74.

Forgy, C. L., 1989. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: J. Mylopoulos & M. L. Brodie, eds. *Readings in Artificial Intelligence and Databases.* San Mateo, CA: Morgan Kaufmann Publishers, Inc., pp. 547-559.

Fougeresa, A. & Ostrosi, E., 2018. Intelligent Agents for Feature Modelling in Computer Aided Design. *Journal of Computational Design and Engineering,* 5(1), pp. 19-40.

Foyle, D. C. et al., 1996. Taxiway Navigation and Situation Awareness (T-NASA) System:

Problem, Design Philosophy, and Description of an Integrated Display Suite for Low-Visibility Airport Surface Operations. *SAE Transactions: Journal of Aerospace,* Volume 105, pp. 1411-1418.

Franke, D. W., 1989. *Representing and Acquiring Teleological Descriptions.* Detroit, MI, Association for the Advancement of Artificial Intelligence, pp. 62-67.

Freeman, D., Reeve, S., Robinson, A. & Ehlers, A., 2017. Virtual Reality in the Assessment, Understanding, and Treatment of Mental Health Disorders. *Psychological Medicine,* 47(14), pp. 2393-2400.

Freund, U., Zerbe, V. & Schorcht, G., 1998. *Integrated System Level Simulation Techniques for the Design of Embedded Systems.* Seattle, WA, Technical University of Ilmenau.

Friedman-Hill, E., 2008. *Jess, the Rule Engine for the Java Platform,* Livermore, CA: Sandia National Laboratories.

Gabbard, J., Hix, D. & Swan, E., 1999. User centered design and evaluation of virtual environments. *Computer Graphics and Applications,* 19(6), pp. 51-59.

Gao, Y. & Gu, N., 2009. Complexity, Human Agents, and Architectural Design: A Computational Framework. *Design Principles and Practices: An International Journal,* 3(6), pp. 115-126.

Garrett, N., Thoms, B., Soffer, M. & Ryan, T., 2007. *Extending the Elgg Social Networking System to Enhance the Campus Conversation.* Pasadena, CA, DESRIST, pp. 14-15.

Gero, J. S., 1990. Design Prototypes: A Knowledge Representation Schema for Design. *AI Magazine,* 11(4), pp. 26-36.

Gero, J. S., 1996. Creativity, Emergence and Evolution in Design. *Knowledge-Based Systems,* 9(7), pp. 435-448.

Gero, J. S. & Kannengiesser, U., 2004. The Situated Function-Behaviour-Structure Framework. *Design Studies,* Volume 25, pp. 373-391.

Gertrudix, F. & Gertrudix, M., 2012. A Study of Music Representation Spaces in Virtual Worlds. *Comunicar,* 19(38), pp. 175-81.

Gianacakes, G., 2003. *Breaking the 5-Sigma Barrier with Systems Engineering.* Washington, DC, Wiley, pp. 313-323.

Gill, T. G. & Hevner, A. R., 2011. A Fitness-Utility Model for Design Science Research. In: H. Jain, A. P. Sinha & P. Vitharana, eds. *Proceedings of the 6th International Conference on Design Science Research in Information Systems.* Milwaukee, WI;: Springer, pp. 237-252.

Gill, T. G. & Hevner, A. R., 2013. A Fitness-Utility Model for Design Science Research. *ACM Transactions on Management Information Systems,* 4(2), pp. 5:1-5:24.

Gips, J., 1975. *Shape Grammars and their Uses: Artificial Perception, Shape Generation and Computer Aesthetics.* Basel: Birkhaeuser Verlag.

Godwin, D., Silcott, M. & Ng, D., 2010. C-A5-03: Virtual Data Warehouse (VDW) Architecture and Implementation. *Clinical Medicine & Research,* 8(3-4), pp. 188-188.

Gosling, J. et al., 2015. *The Java Language Specification ,* Redwood City, CA: Oracle America, Incorporated.

Gosling, J. & McGilton, H., 1996. *The Java Language Environment,* Mountain View, CA: Sun Microsystems.

Graziano, C. D., 2011. *A Performance Analysis of Xen and KVM Hypervisors for hosting the Xen Worlds Project,* Ames, IA: Iowa State University.

Green, M., 1995. The Design of Narrative Virtual Environments. In: *Design, Specification and Verification of Interactive Systems.* Vienna: Springer, pp. 279-293.

Gregor, S. & Hevner, A., 2013. Positioning and Presenting Design Science Research for maximum Impact. *Management Information Systems Quaterly,* 37(2), pp. 337-355.

Gregory, S. A. ed., 1966. *The Design Method.* New York, NY: Springer.

Guerin, K. & Hager, G. D., 2016. United States, Patent No. US20160257000.

Gu, N. & Maher, M. L., 2003. A Grammar for the Dynamic Design of Virtual Architecture Using Rational Agents. *International Journal of Architectural Computing,* 1(4), pp. 489-501.

Gu, N. & Maher, M. L., 2004. *Generating Virtual Architecture with Style.* Launceston,

Tasmania, School of Architecture, University of Tasmania, pp. 141-147.

Gu, N. & Maher, M. L., 2005. Dynamic Designs of 3D Virtual Worlds Using Generative Design Agents. In: B. Martens & A. Brown, eds. *Computer Aided Architectural Design Futures.* Dordrecht: Springer, pp. 239-248.

Hall, A., 1996. Using Formal methods to Develop an ATC Information System. *IEEE Software,* 13(2), pp. 66-76.

Hattenhauer, D., 1984. The Rhetoric of Architecture: A Semiotic Approach. *Communication Quaterly,* 32(1), pp. 71-77.

Haywood, J. E., 1995. *Improving the Management of an Air Campaign with Virtual Reality,* Montgomery, AL: The School of Advanced Airpower Studies, Air University, Maxwell AFB.

Hendrikx, M., Meijer, S., Velden, J. V. D. & Iosup, A., 2013. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications and Applications,* 9(1).

Henrysson, A., Billinghurst, M. & Ollila, M., 2005. *Virtual Object Manipulation using a Mobile Phone.* Christ Church, New Zealand, ACM, pp. 164-171.

Heo, M., Kim, N. & Faith, M. S., 2015. Statistical Power as a Function of Cronbach Alpha of Instrument Questionnaire Items. *BMC Medical Research Methodology,* 15(86).

Hevner, A. R., March, S. T., Park, J. & Ram, S., 2004. Design Science in Information Systems Research. *MIS Quarterly,* 28(1), pp. 75-105.

Higuera-Toledano, M. T., 2017. Java Technologies for Cyber-Physical Systems. *Transactions on Industrial Informatics,* 13(2), pp. 680-687.

Hoisl, F. & Shea, K., 2013. Three-Dimensional Labels: A Unified Approach to Labels for a General Spatial Grammar Interpreter. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing.* New York, NY: Cambridge University Press, pp. 359-375.

Holley, D., Hobbs, M., Howlett, P. & Sawyerr, W., 2013. 'The Chaotic Science Lab': Supporting Trainee Science Teachers - a Cross-Departmental Project. *Networks,* Issue 16, pp. 51-57.

Hong, G., 2015. *Concepts and Modelling Techniques for Pervasive and Social Games,* Trondheim: Department of Computer and Information Science, Norwegian University of Science and Technology.

Huppatz, D. J., 2015. Revisiting Herbert Simon's "Science of Design". *Design Issues,* 31(2), pp. 29-40.

Hyndman, A., 2011. *Method and Apparatus for Controlling a Camera View into a Three Dimensional Computer-Generated Virtual Environment.* United States, Patent No. 20110227913 A1.

Isdale, J., 1993. *What Is Virtual Reality? A Homebrew Introduction and Information Resource List.* [Online] Available at: http://www.columbia.edu/~rk35/vr/vr.html [Accessed 22 December 2016].

Jacobson, I., Christerson, M., Jonsson, P. & Oevergaard, G., 1993. *Object-Oriented Software Engineering.* 4th ed. Wokingham: Addison-Wesley.

Jahn, D., 2014. The Three Worlds of Environmental Politics. In: A. Duit, ed. *State and Environment: The Comparative Study of Environmental Governance.* Cambridge, MA: The MIT Press, pp. 81-110.

Janelle, D. G. & Hodge, D. C. eds., 2000. *Information, Place, Cyberspace, and Accessibility: Issues in Accessibility.* Berlin: Springer-Verlag.

Johnston, R., Melwani, V., Mortimer, S. & Shimura, Y., 2017. *Methods and/or Systems for Designing Virtual Environments.* United States, Patent No. US9814983B2.

Kalay, Y. E. & Marx, J., 2001. The Role of Place in Cyberspace. In: H. Thwaites & L. Addison, eds. *Proceedings of the 7th International Conference on Virtual Systems and Multimedia.* Berkeley, CA: IEEE, pp. 770-779.

Kalay, Y. E. & Marx, J., 2003. Changing the Metaphor: Cyberspace as a Place. In: *Proceedings of the 10th International Conference on Computer Aided Architectural Design Futures.* Tainan: CumInCAD, pp. 19-28.

Kalay, Y. E. & Marx, J., 2005. Architecture and the Internet: Designing Places in Cyberspace. *First Monday,* Issue 5.

Kaur, K., 1997. Designing virtual environments for usability. *Proceedings on Human-*

Computer Interaction, pp. 636-639.

Knight, T., 2000. *Shape Grammars in Education and Practice: History and Prospects.* [Online] Available at: http://www.mit.edu/~tknight/IJDC/ [Accessed 04 February 2016].

Knight, T. W., 1999. *Applications in Architectural Design, and Education and Practice: Report for NSF/MIT Workshop on Shape Computation,* Cambridge, MA: CumInCAD.

Kruchten, P., 2000. From Waterfall to Iterative Development -- A Challenging Transition for Project Managers. *The Rational Edge*, December.

Lefebvre, H., 1991. *The Production of Space.* 1st ed. Oxford: Blackwell.

Livari, J., 2007. A Paradigmatic Analysis of Information Systems As a Design Science. *Scandinavian Journal of Information Systems,* 19(2), pp. 39-64.

Lopes, C. V., 2011. Hypergrid: Architecture and Protocol for Virtual World Interoperability. *Internet Computing,* 15(5), pp. 22-29.

Madni, A. M., 2018. From Models to Stories. In: *Transdisciplinary Systems Engineering: Exploiting Convergence in a Hyper-Connected World.* Cham: Springer International Publishing AG, pp. 59-87.

Maentymaeki, M. & Riemer, K., 2014. Digital Natives in Social Virtual Worlds: A Multi-Method Study of Gratifications and Social Influences in Habbo Hotel. *International Journal of Information Management,* 34(2), pp. 210-220.

Maher, M. L. & Gero, J. S., 2002. *Agent Models of 3D Virtual Worlds.* Pomona, CA, California State Polytechnic University.

Maher, M. L., Gero, J. S., Smith, G. & Gu, N., 2003. Cognitive Agents in 3D Virtual Worlds. *International Journal of Design Computing,* Volume 6.

Maher, M. L. & Gu, N., 2003. Situated Design of Virtual Worlds Using Rational Agents. In: *Proceedings of the Second International Conference on Entertainment Computing.* Pittsburgh, PA: ACM, pp. 1-9.

Maher, M. L., Gu, N. & Li, F., 2001. Visualisation and Object Design in Virtual

Architecture. In: *Proceedings of the Sixth Conference on Computer Aided Architectural Design Research in Asia.* Sydney: s.n., pp. 39-50.

Maher, M. L., Simoff, S., Gu, N. & Lau, K. H., 2000. Designing Virtual Architecture. In: *Proceedings of CAADRIA.* Singapore: s.n., pp. 481-490.

Maher, M. L., Smith, G. & Gero, J. S., 2003. *Designing 3D Virtual Worlds as a Society of Agents.* Dordrecht, Kluwer Academic Publishers.

Maher, M. L., Smith, G. J. & Gero, J. S., 2003. *Design Agents in 3D Virtual Worlds.* Acapulco, ACM.

Maldovan, K. D., Messner, J. I. & Faddoul, M., 2006. Framework for Reviewing Mockups in an Immersive Environment. *Proceedings of CONVR*.

Mallempati, V., Lyle, R. D. & Jones, A. R., 2010. *Method for Virtual World Event Notification.* United States, Patent No. 20100005480 A1.

March, S. T. & Smith, G. F., 1995. Design and Natural Science Research on Information Technology. *Decision Support Systems,* 15(4), pp. 251-266.

March, S. T. & Storey, V. C., 2008. Design Science in the Information Systems Discipline: An Introduction to the Special Issue on Design Science Research. *MIS Quarterly,* 32(4), pp. 725-730.

Marheineke, M., 2016. *Designing Boundary Objects for Virtual Collaboration.* Wiesbaden: Springer.

Martinez, A. D., 2016. *Model-Based Interface Framework : An Extensible Framework for Automatic Generation of Test Cases and Wrappers,* Eindhoven: Technical University of Eindhoven.

McKay, J., Marshall, P. & Hirschheim, R., 2012. The Design Construct in Information Systems Design Science. *Journal of Infromation Technology,* Volume 27, pp. 125-139.

Merrifield, A., 1993. Place and Space: A Lefebvrian Reconciliation. *Transactions of the Institute of British Geographers,* 18(4), pp. 516-531.

Milgram, P. & Kishino, F., 1994. A Taxonomy of Mixed Reality Visual Displays. *Transactions on Information Systems,* 77(12), pp. 1321-1329.

Milgram, P., Takemura, H., Utsumi, A. & Kishino, F., 1994. *Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum.* Boston, MA, SPIE, pp. 282-292.

Minocha, S. & Reeves, A. J., 2010. Interaction Design and Usability of Learning Spaces in 3D Multi-User Virtual Worlds. *Human Work Interaction Design: Usability in Social, Cultural and Organizational Contexts*, pp. 157-167.

Moggridge, B., 2007. *Designing Interactions.* Cambridge, MA: The MIT Press.

Molina, J. P. et al., 2003. *Bridging the gap: developing 2D and 3D user interfaces with the IDEAS methodology.* Berlin, Springer, pp. 303-315.

Mor, Y., 2010. *A Design Approach to Research in Technology Enhanced Mathematics Education,* London: Institute of Education, University of London.

Moynihan, G. P., Ray, P. S., Batson, R. G. & Nichols, W. G., 2001. Application of Total Quality Management Techniques to Safety Analysis in Software Product Development. *International Journal of Technology Management,* 21(3-4).

Mustafa, M. I., Sjostrom, J. & Lundstrom, J. E., 2014. An Empirical Account of Fitness-Utility: A Case of Radical Change towards Mobility in DSR Practice. In: M. C. Tremblay, et al. eds. *Proceedings of the 9th International Conference on Design Science Research in Information Systems.* Miami, FL: Springer, pp. 289-303.

Nagpal, R., Mehrotra, D. & Bhatia, P. K., 2017. The State of Art in Website Usability Evaluation Methods. In: S. Saeed, Y. A. Bamarouf, T. Ramayah & S. Z. Iqbal, eds. *Design Solutions for User-Centric Information Systems.* Hershey, PA: IGI Global, pp. 275-296.

Nerlich, G., 1994. *The Shape of Space.* 2nd ed. Cambridge: Press Syndics of the University of Cambridge.

Nielsen, J., 1992. The Usability Engineering Lifecycle. *Computer,* 25(3), pp. 12-22.

Nielsen, J., 1993. Iterative User Interface Design. *IEEE Computer,* 26(11), pp. 32-41.

Nielsen, J. & Molich, R., 1990. *Heuristic Evaluation of User Interfaces.* Seattle, WA, ACM, pp. 249-256.

Nulty, D. D., 2008. The Adequacy of Response Rates to Online and Paper Surveys: What can be done?. *Assessment & Evaluation in Higher Education,* 33(3), pp. 301-314.

Obeysekare, U. et al., 1996. *Virtual Workbench - A Non-Immersive Virtual Environment for Visualizing and Interacting with 3D Objects for Scientific Visualization.* Los Alamitos, CA, IEEE Computer Society Press, pp. 345-500.

Offermann, P., Levina, O., Schoenherr, M. & Bub, U., 2009. Outline of a Design Science Research Process. In: *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology.* Philadelphia, PA: ACM.

Oliveira, E., Simoes, F. P. M. & Correia, W. F., 2017. Heuristics Evaluation and Improvements for Low-Cost Virtual Reality. In: F. L. S. Nunes & J. C. D. Oliveira, eds. *Proceedings of the 19th Symposium on Virtual and Augmented Reality.* Parana: IEEE, pp. 178-187.

Oliveira, M., Crowcroft, J. & Slater, M., 2003. *An Innovative Design Approach to Build Virtual Environment Systems.* Zurich, ACM, pp. 143-151.

OpenSimulator, 2017. *Configuration.* [Online] Available at: http://opensimulator.org/wiki/Configuration [Accessed 04 December 2017].

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S., 2008. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems,* 24(3), pp. 45-77.

Pellas, N., Kazanidis, I., Konstantinou, N. & Georgiou, G., 2017. Exploring the Educational Potential of Three-Dimensional Multi-User Virtual Worlds for STEM Education: A Mixed-Method Systematic Literature Review. *Education and Information Technologies,* 22(5), p. 2235–2279.

Polcar, J., Gregor, M., Horejsi, P. & Kopecek, P., 2016. *Methodology for Designing Virtual Reality Applications.* Vienna, DAAAM International, pp. 768-774.

Poole, D. L. & Mackworth, A. K., 2010. *Artificial Intelligence: Foundations of Computational Agents.* Cambridge: Cambridge University Press.

Poole, D., Mackworth, A. & Goebel, R., 1998. *Computational Intelligence: A Logical Approach.* New York, NY: Oxford University Press.

Poppendieck, M. & Poppendieck, T., 2003. *Lean Software Development: An Agile Toolkit.*

Upper Saddle River, NJ: Pearson Education.

Popper, K., 1979. *Objective Knowledge: An Evolutionary Approach.* 2nd ed. Oxford: Oxford University Press.

Ralph, P. & Wand, Y., 2008. A Teleological Process Theory of Software Development. *Sprouts: Working Papers on Information Systems,* 8(23).

Ralph, P. & Wand, Y., 2009. A Proposal for a Formal Definition of the Design Concept. In: K. Lyytinen, P. Loucopoulos, J. Mylopoulos & B. Robinson, eds. *Design Requirements Engineering: A Ten-Year Perspective.* Berlin: Springer-Verlag, pp. 103-136.

Rao, I. & Prasad, R., 2012. Software Engineering for Different Viewpoints Approach to Requirements Engineering. *International Journal of Computers Electrical and Advanced Communications Engineering,* 1(1), pp. 149-166.

Reis, R., Escudeiro, P. & Fonseca, B., 2012. *High-Level Model for Educational Collaborative Virtual Environments Development.* Rome, IEEE, pp. 356-358.

Ritsos, P. D. et al., 2013. Training Interpreters using Virtual Worlds. In: M. L. Gavrilova, C. J. K. Tan & A. Kuijper, eds. *Transactions on Computational Science XVIII.* Berlin: Springer-Verlag, pp. 21-40.

Rittel, H. W. J., 1987. *The Reasoning of Designers,* Stuttgart: IGP.

Rosenberg, L. B., 1992. *The Use of Virtual Fixtures as Perceptual Overlays to Enhance Operator Performance in Remote Environments,* Stanford, CA: Center for Design Research, Standford University.

Rosenman, M. A. & Gero, J. S., 1998. Purpose and Function in Design: From the Socio-Cultural to the Techno-Physical. *Design Studies,* 19(2), pp. 161-186.

Roussou, M. et al., 2004. *A User-Centered Approach on Combining Realism and Interactivity in Virtual Environments.* Chicago, IL, IEEE.

Roy, U. et al., 2001. Function-to-Form Mapping: Model, Representation and Applications in Design Synthesis. *Computer-Aided Design,* 33(10), pp. 699-719.

Ruddle, R. A. & Peruch, P., 2004. Effects of Proprioceptive Feedback and Environmental

Characteristics on Spatial Learning in Virtual Environments. *International Journal of Human-Computer Studies,* 60(3), pp. 299-326.

Rus-Calafell, M. et al., 2015. Confronting Auditory Hallucinations Using Virtual Reality: The Avatar Therapy. *Studies in Health Technology and Informatics,* Volume 219, pp. 192-196.

Russell, S. & Norvig, P., 2010. *Artificial Intelligence: A Modern Approach.* 3rd ed. Upper Saddle River, NJ: Pearson Education.

Savin-Baden, M., Falconer, L., Wimpenny, K. & Callaghan, M., 2017. Virtual Worlds for Learning. In: E. Duval, M. Sharples & R. Sutherland, eds. *Technology Enhanced Learning.* Cham: Springer, pp. 97-107.

Sawyerr, W., 2016. An Application of the Design Science Research Theoretical Framework and Methodology in the Construction of an Approach for Designing Applications in 3D Virtual Worlds. *Journal of Advances in Information Technology,* 7(4), pp. 259-263.

Sawyerr, W., 2016. An Approach for Designing Applications in 3D Virtual Worlds. In: J. Botev, ed. *ACM (Association for Computing Machinery), 8th International Workshop on Massively Multiuser Virtual Environments.* Klagenfurt: ACM, pp. 9-10.

Sawyerr, W. A. & Pinkwart, N., 2011. Designing a 3D Virtual World for Providing Social Support for the Elderly. In: *ECSCW (European Conference on Computer Supported Cooperative Work), Workshop on Fostering Social Interactions in the Ageing Society.* Aarhus: Department of Infomatics, Humboldt University of Berlin.

Sawyerr, W. A. & Pinkwart, N., 2011. Extending the Private and Domestic Spaces of the Elderly. In: J. Rojas, Y. Theng & N. Pang, eds. *CSCW (Computer Supported Cooperative Work), Workshop on Socialising Technology Among Seniors in Asia.* Hangzhou: ACM, pp. 867-868.

Sawyerr, W., Brown, E. & Hobbs, M., 2013. Using a Hybrid Method to Evaluate the Usabilityof a 3D Virtual World User Interface. *International Journal of Information Technology & Computer Science,* 8(2), pp. 66-74.

Sawyerr, W. & Hobbs, M., 2014. Designing for Usability in 3D Virtual Environments. In: S. R. Humayoun, et al. eds. *IFIP (International Federation for Information Processing), 2nd International Workshop on Usability and Accessibility Focused Requirements Engineering.* Karlskrona: IEEE, pp. 32-35.

Schaller, T. W., 1997. *The Art of Architectural Drawing.* New York, NY: Van Nostrand Reinhold.

Schatten, M., Duric, B. O., Tomicic, I. & Ivkovic, N., 2017. Agents as Bots - An Initial Attempt Towards Model-Driven MMORPG Gameplay. In: Y. Demazeau, P. Davidsson, J. Bajo & Z. Vale, eds. *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection.* Porto: Springer, pp. 246-258.

Schooten, B. v., Donk, O. & Zwiers, J., 1999. *Modelling Interaction in Virtual Environments using Process Algebra.* Enschede, University of Twente, pp. 195-212.

Scott, E., Soria, A. & Campo, M., 2017. Adaptive 3D Virtual Learning Environments - A Review of the Literature. *Transactions on Learning Technologies,* 10(3), pp. 262-276.

Sedrez, M. R. & Periera, A. T. C., 2012. Fractal Shape. *Nexus Network Journal,* 14(1), pp. 97-107.

Sharp, H., Rogers, Y. & Preece, J., 2007. *Interaction Design: Beyond Human-Computer Interaction.* 2nd ed. Chichester: John Wiley & Sons.

Shaw, M., 1995. Comparing Architectural Design Styles. *Software,* 12(6), pp. 27-41.

Sherstyuk, A. & Treskunov, A., 2014. *Space-Time Maps for Virtual Environments.* Bremen, The Eurographics Association, pp. 45-48.

Shore, J. & Warden, S., 2008. *The Art of Agile Development.* Sebastopol, CA: O'Reilly Media.

Silver, M. S., Markus, M. L. & Beath, C. M., 1995. The Information Technology Interaction Model: A Foundation of the MBA Core Course. *MIS Quaterly,* 19(3), pp. 361-390.

Simon, H. A., 1962. The Architecture of Complexity. In: G. W. Corner, ed. *Proceedings of*

the American Philosophical Society. Philadelphia, PA;: American Philosophical Society, pp. 467-482.

Simon, H. A., 1996. *The Sciences of the Artificial.* 3rd ed. Cambridge, MA: The MIT Press.

Singh, V. & Gu, N., 2012. Towards an Integrated Generative Design Framework. *Design Studies,* Volume 33, pp. 185-207.

Smith, B. L., 2012. *3ds Max Design Architectural Visualization.* Waltham, MA: Focal Press.

Soja, E. W., 1985. The Spatiality of Social Life: Towards a Transformative Retheorisation. In: D. Gregory & J. Urry, eds. *Social Relations and Spatial Structures.* London: Macmillan Education UK, pp. 90-127.

Sommerville, I., Sawyer, P. & Viller, S., 1998. Viewpoints for Requirements Elicitation: A Practical Approach. In: *Proceedings of the 3rd International Conference on Requirements Engineering.* Colorado Springs, CO: IEEE, pp. 74-81.

Srinivasan, V., Mandal, E. & Akleman, E., 2005. *Solidifying Wireframes.* Alberta, Central Plain Book Manufacturing, pp. 203-210.

Stauffer, L. A., Diteman, M. & Hyde, R., 1991. Eliciting and Analysing Subjective Data about Engineering Design. *Journal of Engineering Design,* 2(4), pp. 351-366.

Stebbing, P. D., 2004. A Universal Grammar for Visual Composition?. *Leonardo,* 37(1), pp. 63-70.

Steinberger, M. et al., 2014. On-the-fly Generation and Rendering of Infinite Cities on the GPU. *Computer Graphics Forum,* 33(2), pp. 105-114.

Steino, N., 2010. Parametric Thinking in Urban Design. In: A. Bennadji, B. Sidawi & R. Reffat, eds. *Proceedings of the Fifth International Conference of the Arab Society for Computer Aided Architectural Design.* Fez: The Arab Society for Computer Aided Architectural Design, pp. 261-270.

Sterelny, K., 2012. From Fitness to Utility. In: S. Okasha & K. Binmore, eds. *Evolution and Rationality: Decisions, Co-operation and Strategic Behaviour.* Cambridge: Cambridge University Press, pp. 246-273.

Stiny, G., 1980. Introduction to Shape and Shape Grammars. *Environment and Planning*

B, Volume 7, pp. 343-351.

Stiny, G. & Gips, J., 1972. *Shape Grammars and the Generative Specification of Painting and Sculpture.* Amsterdam, IFIP, pp. 1460-1465.

Street Kitchen, 2017. *Our Locations.* [Online] Available at: http://www.streetkitchen.co.uk/locations/airstream-two [Accessed 08 April 2018].

Sutcliffe, A. G. et al., 2018. Reflecting on the Design Process for Virtual Reality Applications. *International Journal of Human-Computer Interaction,* pp. 1-12.

Sylvain, D., Sauriol, N. & Hyndman, A., 2010. *Method and Apparatus for Managing Communication Between Participants in a Virtual Environment.* United States, Patent No. 7840668 B1.

Tait, A., 1992. *Desktop Virtual Reality.* London, IET, pp. 1-5.

Takeda, H., Veerkamp, P., Tomiyama, T. & Yoshikawam, H., 1990. Modeling Design Processes. *AI Magazine,* pp. 37-48.

Tall, D., 2004. *Thinking Through Three Worlds of Mathematics.* Bergen, Bergen University College, pp. 281-288.

Tarkan, S., 2009. *The Formal Specification of a Kitchen Environment,* College Park, MD: University of Maryland.

Tavakol, M. & Dennick, R., 2011. Making Sense of Cronbach's Alpha. *International Journal of Medical Education,* Volume 2, pp. 53-55.

Tepavcevic, B., 2017. Design Thinking Models for Architectural Education. *The Journal of Public Space,* 2(3), pp. 67-72.

The Center for the Study of Art and Architecture, 2002. *Elements of Architectural Design: Sensory Shape.* [Online] Available at: http://www.architeacher.org/aesthetics/archi-elements1c.html [Accessed 03 June 2018].

Tillich, P., 1987. *On Art and Architecture.* 1st ed. New York City, NY: Crossroad.

Tirpak, T. M. & Ramic, H., 2014. *Method of Providing a Shared Virtual Lounge Experience.* United States, Patent No. 8812358 B2.

Tizani, W., 2011. Collaborative Design in Virtual Environments at Conceptual Stage. In: X.

Wang & T. J. J, eds. *Collaborative Design in Virtual Environments.* Berlin: Springer, pp. 67-76.

Tse, T. H. & Pong, L., 1989. Towards a Formal Foundation for DeMarco Data Flow Diagrams. *The Computer Journal,* 32(1), pp. 1-12.

Tur, J. Y., 2005. *L'Hemisfèric.* [Art] (Ciudad de las Artes y las Ciencias).

Twidale, M., Rodden, T. & Sommerville, I., 1993. The Designers' Notepad: Supporting and Understanding Cooperative Design. In: G. d. Michelis, C. Simone & K. Schmidt, eds. *Proceedings of the Third European Conference on Computer-Supported Cooperative Work.* Milan, Italy: Springer, pp. 93-108.

Ullrich, C. J. & Kunkler, K. J., 2018. *Virtual Tool Manipulation System.* United States of America, Patent No. US9881520B2.

Van Dam, A. et al., 2000. Immersive VR for Scientific Visualization: A Progress Report. *Computer Graphics and Applications,* 20(6), pp. 26-52.

Vanacken, L., Grossman, T. & Coninx, K., 2007. Exploring the Effects of Environment Density and Target Visibility on Object Selection in 3D Virtual Environments. In: W. Steurzlinger, Y. Kitamura & S. Coquillart, eds. *Symposium on 3D User Interfaces.* Charlotte, NC: IEEE, pp. 115-122.

Venable, J. R., 2010. Design Science Research Post Hevner et al.: Criteria, Standards, Guidelines, and Expectations. In: R. Winter, J. L. Zhao & S. Aier, eds. *Global Perspectives on Design Science Research.* Berlin: Springer, pp. 109-123.

Venkatasubramanian, V., 2007. A theory of design of complex teleological systems: Unifying the Darwinian and Boltzmannian perspectives. *Complexity,* 12(3), pp. 14-21.

Vince, J., 1998. *Essential Virtual Reality fast: How to Understand the Techniques and Potential of Virtual Reality.* London: Springer London.

Vukicevic, V., Jones, B., Gilbert, K. & Wiemeersch, C. V., 2016. *WebVR.* [Online] Available at: https://w3c.github.io/webvr/ [Accessed 19 December 2016].

Walls, J., Widmeyer, G. & El Sawy, O., 1992. Building an Information System Design Theory for Vigilant EIS. *Information Systems Research,* 3(1), pp. 36-59.

Ward, P. & Mellor, S., 1985. *Structured Development for Real-Time Systems.* Englewood Cliffs, NJ: Prentice Hall.

Weeden, K. A. & Grusky, D. B., 2012. The Three Worlds of Inequality. *Journal of Sociology,* 117(6), pp. 1723-1785.

Wharton, C., Rieman, J., Lewis, C. & Polson, P., 1994. *The Cognitive Walkthrough Method: A Practitioner's Guide.* New York, NY: John Wiley & Sons.

Whisker, V. E. et al., 2003. Using immersive virtual environments to develop and visualize construction schedules for advanced nuclear power plants. *Proceedings of ICAPP*, pp. 4-7.

White, B., 2008. *A Bridge between Virtual Worlds: Second Life's new program links Virtual Environments.* [Online] Available at: https://www.technologyreview.com/s/410578/a-bridge-between-virtual-worlds/ [Accessed 10 January 2018].

Winterbottom, C., Gain, J. & and Blake, E., 2005. *Experiences in Designing a User-Oriented Tool for Building and Understanding Interactions in Virtual Environments,* Cape Town: Collaborative Visual Computing Laboratory, Department of Computer Science, University of Cape Town.

Witmer, B. G., Bailey, J. H., Knerr, B. W. & Parsons, K. C., 1996. Virtual Spaces and Real World Places: Transfer of Route Knowledge. *International Journal of Human-Computer Studies,* 45(4), pp. 413-428.

Wooldridge, M., 2009. *An Introduction to MultiAgent Systems.* 2nd ed. Chichester: John Wiley & Sons.

Wordsworth, J., 1996. *Software Engineering with B.* Wokingham: Addison-Wesley.

Yan, H. & Damian, P., 2008. *benefits and Barriers of Building Information Modellling.* Beijing, Tingshua University Press.

Yu, R., Gu, N. & Ostwald, M., 2012. Using Situated FBS Ontology to Explore Designers' Patterns of Behavior in Parametric Environments. *Journal of Information Technology in Construction,* Volume 17, pp. 271-282.

Zhou, Y., Murata, T. & DeFanti, T. A., 2000. Modeling and Performance Analysis using

Extended Fuzzy-Timing Petri Nets fro Networked Virtual Environments. *Transactions on Systems, Man, and Cybernetics, Part B,* 30(5), pp. 737-756.

Zhu, Y. et al., 2016. *SAVE: Shared Augmented Virtual Environment for Real-Time Mixed Reality Applications.* Zhuhai, China, ACM, pp. 13-21.

Zuniga, F., Ramos, F. & Piza, H. I., 2005. Specifying Agent's Goals in 3D Scenarios Using Process Algebras. In: F. F. Ramos, V. Larios Rosillo & U. H, eds. *Advanced Distributed Systems.* Berlin: Springer, pp. 472-482.

Appendix A. Extended Cognitive Walkthrough

A. Task action

1) Form goal

a) Can the user form or remember the task goal?

The task knowledge required for this stage was simple and it was therefore concluded that the user is able to determine and remember what to do. In case the user has poor task knowledge, the application provided features that include help tips and a main interactive signpost about the task and how to complete it. One of the problems found at this stage is related to navigation within the application (see: Navigation cycle B.1.a).

b) Can the user form an intention of what to do?

Besides the hints given at the start about what to do, the application did not suggest the best course of action. This is in part due to the nature of the application in that it provides results of the user's try at the end. However, it was expected that trying to put objects in obviously incorrect locations (e.g. placing a conical flask on a laptop as its final destination) will generate some sort of feedback response.

2) Locate active environment

a) Are the appropriate objects or parts of the environment visible?

The environment was highly detailed with very good graphical content producing realistic imagery. Also most of the objects that were necessary to locate in order to carry out the task action were visible. However, some of the objects that were made of glass material rendered poorly. Their low contrast against the environment of the virtual lab made them a bit difficult to see.

3) Locate objects

a) Can the necessary objects be located?

As mentioned in A.2.a there were problems with correctly rendering glass-made objects within the application. Low contrast of these objects made them difficult to see and because of this they were also difficult to find. Nonetheless, the objects were still discernible and with a bit of effort the user is able to locate them.

4) Approach and orient

a) Can the user approach and orient themselves to carry out the necessary action?

Here no problems were encountered in terms of approaching and orientation. However, with regards to missing generic design properties, the user's navigation could be guided in this action stage by providing hints within the application (see: Navigation cycle B.1.a).

5) Specific action

a) Can the user decide what action to take and how?

Within the given task scenario the user follows a goal-directed mode of exploration. Therefore task knowledge determines what actions the user should take.

6) Manipulate object

a) Can the user carry out the manipulation easily?

The user is able to carry out the manipulation easily as there are no complicated virtual tools required to handle the objects.

7) Recognise feedback

a) Is the consequence of action visible?

Sufficient and recognisable feedback is provided to the user as a consequence of their action. For example when the user clicks on an object to be moved, an action dialogue box helps the user complete the move. Once the move has been completed, a message is displayed informing the user that the new position has been recorded.

8) Assess feedback

a) Can the user interpret the change?

Feedback is clear and unambiguous hence the user is able to interpret the change as he or she manipulates the objects in the application.

9) Specify next action

a) Can the user decide what to do next?

Our test included a single scenario with continuous action. In this context, the user is always within procedure and relies mainly on task knowledge to carry out all the steps necessary to complete the task goal.

B. Navigation

1) Find path to target

a) Does the user know where to start looking?

The task knowledge required for the scenario was clear and simple. However, navigation towards target objects is mostly by free-form movement rather than directed or rather aided pathways. The only way the user knows where to start looking or where to locate the target is by looking everywhere.

2) Decide direction

a) Can the user determine a pathway towards the target?

The user is unable to determine an exact pathway towards the target objects – there are clues concerning what to do, but there are none as to where they might be located (e.g. a mini-map overlay with guidance to the general area an object is located or active/interactive cues leading to the targets).

3) Move navigate

a) Can the user execute movement and navigation actions?

Movement and navigation are fairly simple and mostly accomplished by intuition. Where this is not the case, Second Life provides a welcome area with basic skills and interaction tutorials for new users.

4) Interpret chance

a) Can the user recognise the search target?

Given the context of the application and the scenario, task knowledge allows the user to recognise most of the search targets as being a hazard or out of place.

5) Record/memorise location

a) Do the facilities for recording locations exist?

There are no waymarking or pathway tracing features built into the application but facilities do exist for recording object locations.

Appendix B. VW Heuristics

A. Design and Aesthetics Heuristics:

(H1) Feedback: A VW must always keep the user informed about the condition of its avatar, related events, or relevant facts that occur inside the VW. The virtual world must provide to the user, easily noticeable feedback related to any action that he begins or affects him/her in a direct or indirect way.

(H2) Clarity: A VW must have a control panel which is easy to understand, and uses a clear language. Also, the elements of the control panel should be shown tidy and grouped in a way that allows the user to find what he is looking for in an intuitive way.

(H3) Consistency: The VW must be consistent in all its aspects, in this way the user can predict the results of every fulfilled action.

(H4) Simplicity: The control panel of the VW should not be overloaded and it must have only the needed and relevant information. The icons, the messages of the system and the interaction with the objects inside the VW must be simple and intuitive.

B. Control and Navigation Heuristics:

(H5) Orientation and navigation: A VW must have an intuitive navigation and memorable. Also, must give the user a way to locate it inside and a way to find a determine location.

(H6) Camera control and visualization: A VW must allow the user to determine the level and quality of the texture, visual effects, or objects with a pure esthetic purpose. Also the VW must give the user the control or angle of the camera from where it is visualized.

(H7) Low memory load: A VW must minimize the demands on the user's memory by making the objects, options and actions visible and easily accessible. Also the system must provide the user ways to register or remember places inside the VW that have been visited or that can be of one's interest.

(H8) Avatar's customization: A VW must offer a whole group of predefined avatars, with a specific genre, age, appearance, among other attributes. The VW must allow the user to change the aspects of the avatar, whenever he/she wants.

(H9) Flexibility and efficiency of use: A VW must provide accelerators for common actions, allow the user to define his own accelerators and change other interface options. This allows the advanced users to interact with the VW in a more efficient way.

(H10) Communication between avatars: The interaction inside the VW must be analogous to the real world. It must be done in an easy and intuitive way and should be clear for the emitter and the receptor.

(H11) Sense of ownership: The physical rules of the real world should be maintained in a VW. In the case that the rules could be changed; the VW must inform these variations in a clear and explicit way.

(H12) Interaction with the Virtual World: A VW must clearly indicate to the users which objects of the VW they can interact with and the ones they cannot, also indicating which actions can be carried out with the objects that they can interact.

C. Errors and Help Heuristics:

(H13) Support to learning: The complex objects of a VW must be complemented with definitions and indications for its use; this way learning may be promoted.

(H14) Error prevention: The VW must prevent users from making any mistake or creating undesired situations, related with the interface or the VW.

(H15) Helps users to recover from errors: A VW must provide the user with the tools to recover from system errors or any undesired situation when the user cannot recover by himself.

(H16) Help and documentation: A VW must give the user relevant information not only online but also inside the VW. This information must be of easy access and written or spoken in the user's language.

A. Design and Aesthetics

(H1) Feedback:

(H1.1) The system informs the user when it has any object of the VW under his possession (like money, elements, etc.).

(H1.2) The VW informs when a message from other user is received.

(H1.3) The system informs the user if the changes in the system are successful or failed.

(H1.4) The VW informs constantly about what happens in the avatar's environment.

(H1.5) All the actions made by the user must provide an immediate feedback.

(H2) Clarity:

(H2.1) The user can understand the functionality of the interface without needing to try its elements.

(H2.2) The user understands the elements of the interface. (H2.3) When the user needs to choose among different

options, the alternatives are not ambiguous.

(H3) Consistency:

(H3.1) All the elements of the interface, besides the dialogues of the NPCs (Non-Player Characters) can be found in the same language.

(H3.2) The interaction with the entire VW is carried out in a similar way.

(H3.3) The user is able to predict the result of the carried out actions.

(H4) Simplicity:

(H4.1) The user interface uses screen space which is significantly small compared with the one used for the display of the VW.

(H4.2) Each element of the interface is clearly distinguished from the others and is easy for the users to identify.

(H4.3) The overload of information is avoided in the VW.

B. Control and Navigation

(H5) Orientation and navigation:

(H5.1) The world offers information for orientation purposes.

(H5.2) There is a VW general map, which any user can access at any time.

(H5.3) The avatar has the possibility to transport himself into the VW.

(H6) Control camera and visualization:

(H6.1) There is an option to hide/show the non-relevant elements in the VW.

(H6.2) The system allows adjusting the execution velocity and graphic quality ratio.

(H6.3) There is a balance between the control panel and the interface.

(H6.4) The system allows the user to change the angles from where the VW is being observed.

(H6.5) The system allows the user to change the distance from where the VW is being observed.

(H6.6) Camera changes don't affect the user's possibility to interact.

(H7) Low memory load:

(H7.1) Frequent options are visible in the interface.

(H7.2) There is no element in the interface that is accessible with more than three levels of depth.

(H7.3) Notices are short and not ambiguous.

(H8) Avatar's customization:

(H8.1) The system allows the user to change the aspect of his avatar without using external elements as ornaments, clothes, and others.

(H8.2) The system also allows the user to change the aspect of his avatar using external elements as ornaments, clothes, and others.

(H8.3) The system allows the user to change the aspect of his avatar at any time.

(H9) Flexibility and efficiency of use:

(H9.1) The system has accelerators for frequents actions.

(H9.2) The system allows associating actions and functions to accelerators.

(H9.3) The user has control over the interface.

(H10) Communication between avatars:

(H10.1) The VW allows communication between two or more avatars.

(H10.2) The VW allows the avatars to communicate through different ways, including written, verbal, facial expression and others.

(H10.3) The VW allows clear identification of the emitter of the messages.

(H11) Sense of ownership:

(H11.1) Avatars are capable of making every “real” move you want (run, walk, jump, look around, and more).

(H11.2) Additional movements that an avatar can do in the VW are informed.

(H11.3) The system informs directly to the users every modification of physical rules that affects the VW.

(H12) Interaction with the Virtual World:

(H12.1) The objects that you can interact with in the VW are easily distinguished from the ones that you cannot.

(H12.2) Once an object is selected in the VW, the user is informed of all the means to interact with it.

(H12.3) Activities in the VW follow a defined logic.

C. Errors and Help

(H13) Learning support:

(H13.1) The system gives means for the user to learn how to interact with it.

(H13.2) The system explains to inexperienced users how the interaction is accomplished in the VW.

(H13.3) The VW offers guided tours for new visitors.

(H14) Error prevention:

(H14.1) There are no elements of the system interface that can confuse the user and lead him to a mistake.

(H14.2) There are no elements of the VW that can be complex, unclear or been hidden, that can lead the user to make a mistake.

(H14.3) The navigation in the VW is intuitive and easy to recall.

(H15) Helps users to recover from errors:

(H15.1) The system gives the user adequate tools to identify and diagnose problems, as error clear messages and information about the origin of the problem.

(H15.2) There are means for the user to get help as needed.

(H15.3) The VW proposed help is context-sensitive.

(H16) Help documentation:

(H16.1) There is basic documentation that can be accessed at any time from inside the VW.

(H16.2) There is clear and complete documentation that can be accessed at any time from outside the VW

(H16.3) The help is clear, straight and given in the user's language.

Appendix C. User Guide for the VWADM's AW Agent Package

User Guide for
the Virtual World Application Design Method's
Active Worlds Agent Package

William Sawyerr
Department of Computing and Technology
Anglia Ruskin University

Contents

1 Overview

This document describes the Virtual World Application Design Method's (VWADM's) Active Worlds (AW) Agent Package using java sensors and effectors and the Jess language for coding agent reasoning as rules.

The aim of the package is a flexible, object oriented framework in which agents be- come the basis for all elements of a VW in AW. By using agent models, it is possible to design intelligent, interactive places in VWs that exhibit rational behaviours. The VWADM's AW Agent Package is based on a set of abstract classes that form the generic architecture for constructing a system of agents. Central to the framework are `MAS` and `Agents` (Figure 1).

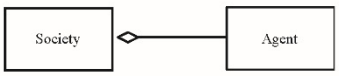


Figure 1: A `MAS` is an aggregation of `Agents`.

A MAS is an aggregation of Agents that share a common connection with a VW. Normally these Agents would share some ontological connection, such as a room agent plus a set of wall agents that collectively comprise a virtual conference room. The MAS manages computational resources, such as the connection to the VW, on behalf of the Agents.

Each entity capable of reflexive, reactive or reflective behaviours is modelled as an agent. The Agent has these generic behaviours and an optional 3D representation, and can both dynamically change the 3D representation plus produce non-visual behaviours. The procedure to assert a 3D object into a VW in AW is to copy an existing object, move it to a desired location, and edit a dialog box to specify its properties. The procedure to create an agent in the VW is to configure the agent as a set of sensors and a rule base. The agents are configured using an XML file that is loaded via a validating parser[1], instead of being statically linked at compile time. This flexibility will

[1]The alternative is to have one agent dynamically load another at runtime.

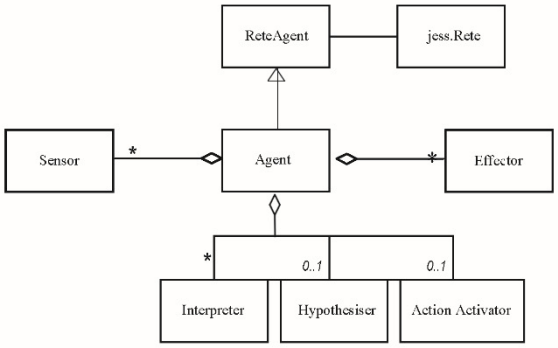


Figure 2: Components comprising an `Agent` or a `ReteAgent`.

eventually provide for the reconfiguration of Agents running in the VW without having to recompile and restart the server.

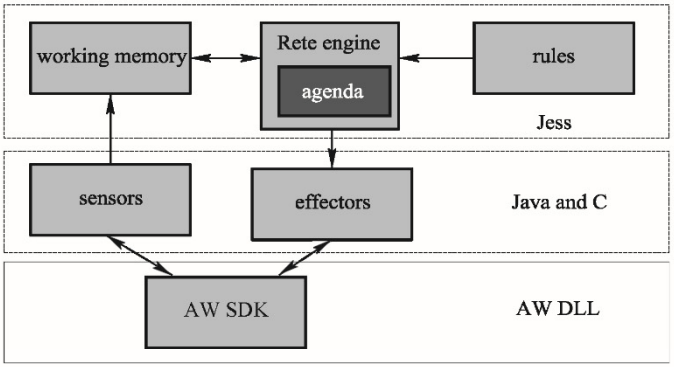


Figure 3: Architecture of `ReteAgent`.

`Sensors` are java objects that sense an agent's environment and `Effectors` act on that environment. These employ a Java Native Interface (JNI) to the AW SDK (Figure 4). An effector is a function call from the right-hand side of a rule. `ReteAgent` is an implementation of an agent that uses Jess for everything except sensors and effectors. A `MAS` can contain one or more agents that are instances of `ReteAgent`. The creator of a `ReteAgent` configures the agent by specifying a set of `Sensors`. Sensor data is stored in the Jess working memory. Messages from other agents are also recorded in Jess' working memory by the sensors. That is, each time new sense data

is received, a new fact (a java bean) is asserted into Jess' working memory. See Sections 5 and 6.

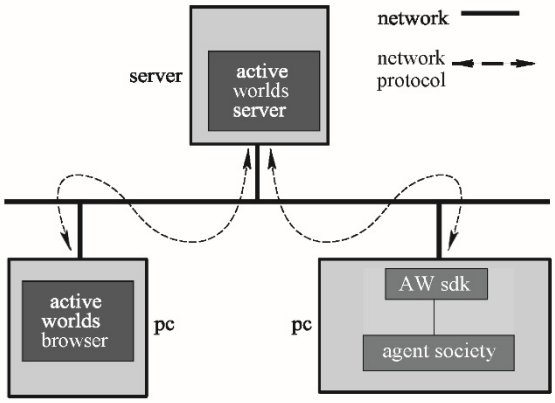


Figure 4: A `MAS` communicates with the AW server via the AW SDK.

`Effector` and `Sensor` classes are written in java. AW sensors and effectors addition ally employ a JNI to the AW SDK. Developers can also write their own sensors and/or effectors by extending the classes in the `vwadm.awa.base` java package.

In order to run the software, you will need at least the Java 1.4 virtual machine (VM) installed. As the AW SDK is a Microsoft Windows DLL, this package will only run under Windows.

2 Files included in the software

The following files comprise the package. The files need to be saved into one directory. In this guide that directory is called `demos`. The jar files contain compiled java[2]. You do not need to unzip these.

- Files needed by all agents (Jess files are not included in this package. You will need to download it yourself. The other files are from base.zip)
  - `base.jar` is the compiled java for the base classes. They are in the documentation as package `vwadm.awa.base`. You don't need to recompile these.
  - `xml-apis.jar` and `xercesImpl.jar` are the DOM 3 XML parser of xerces 2 from www.apache.org.
  - `jess.jar` containing Jess v6.0 or later. Go to `http://herzberg.ca.sandia.gov/jess/` to download version. You can get a full version for free for academic use but

[2]A jar file is a zip format file with an extra manifest added for use by the java VM.

you need to complete a form saying what it is for etc. There is a trial version that has an expiry period.

- `aw.dll` is a JNI library that wraps around the AW SDK.
- `awa.dll` is the SDK from AW.

- Files specific to each agent - for the Section 9 demo (from demos.zip):
  - `newoffice.bat` is an MS-DOS batch file that runs a system of agents "Office" with agent "door". Edit this as per Section 3 in order to run the office demo, or copy and edit to create your own agent.
  - `user.properties` is a file containing java system properties. Edit this as per Section 3 to run the office demo, or copy and edit to create your own agent.
  - `newdoor.clp` contains the Jess rules for agent "door".
  - `newoffice.xml` contains the configuration for MAS "Office".

3 Manual Configuration - Details of files that need to be modified

This section describes how to manually modify the files to configure an agent. The alternative is to use the `configurator` of Sectionconfigurator.

These changes are described assuming that you are running the office demo. To create your own agent you will copy each of the Office files and rename the files to be the name of your agent. Then you will edit these new agent files to have your agent's configuration, properties, rules, etc.

3.1 Edit bat file

The first thing you need to do is to edit the bat file. Assuming that you are running MAS Office, edit `newoffice.bat`.

The first line of `newoffice.bat` sets the `PATH` environment variable so that the operating system can find the java VM, and so that the agent can find the two `dll` files:

```
set PATH=.;C:\jdk1.8.0_112\bin;C:\Windows\System32;C:\Windows
```

Note that there cannot be any whitespace around the = character (this is an MS-DOS thing), and that MD-DOS is not case sensitive. The `PATH` is just a list of directories to look in for the operating system to find files, separated by semicolons - again with no whitespace, and using MS-DOS pathnames not long MS Windows ones. The first one is the directory that you are running the agent from. The second is the bin directory of the java VM. The rest are for the operating system. Each of these needs to be made up of an 8+3 character MS-DOS directory and file names. You can check the default `PATH` for the operating system from the control panel. On Windows 10 you do this:

- In Search, search for and then select: Control Panel
- Click on "System and Security"
- Click on "System"
- Click the "Advanced system settings" link
- Click on the "Advanced" panel
- Click on the "Environment Variables" button

and look for the `PATH` variable under "System variables". You can find the MS-DOS name for a file or directory by opening an MS-DOS command shell, changing to the relevant drive and directory, and typing `dir /X`

The above path assumes that you have the java SE development kit (JDK) 8 or later (from http://www.oracle.com) installed on your machine.

Change the path accordingly if this is not the case.

The second line of `newoffice.bat` is the classpath. This should contain a list of the jar files in your directory.

The last line of `newoffice.bat` runs the program. For example, (this is a *single* line),

```
java -Dvwadm.awa.base.MAS.inFile="newoffice.xml" vwadm.awa.base.MAS
```

runs the MAS specified by the XML configuration `newoffice.xml` in the current directory.

3.2 Edit properties file

Change the following properties in `user.properties`:

- `vwadm.citizen` to your citizen number. This citizen should have BUILD and BOT rights in the VW that the agent is to run in.
- `vwadm.privilegePassword` to your privilege password

3.3 Edit XML file

Change the following parameters in `newoffice.xml`; see
http://mojo.csd.anglia.ac.uk/william/MAS.dtd
for the grammar of the XML

- In element `connection` you need to set `domain`, `port` and `worldName` according to your AW server, set `avatarType` to the number of an avatar, and `avatarname` to any name you like.
- In element `reteagent`, set `name` to any name you like your agent to have.

  Set the parameter with name `jess` to have as `value` the URL of the clp file containing your Jess rules (such as "file:newofficedoor.clp"). A `MAS` can contain any number of `Agent` and/or `reteagent` elements. Any URL can be used for the clp file, such as

``http://mojo.csd.anglia.ac.uk/william/AWbot/demos/newofficedoor.clp''

or

``file:///demo/newofficedoor.clp''

You can add any number of parameters you wish providing you at least provide the `jess` parameter. You then use the `getConfigurationParameter` method on `ReteAgent` to retrieve them from your clp program.

- Set the `location` element to the desired location of your MAS. The x value is positive West direction and 1W is the same as x=1000. The y value is the up direction. The z value is positive North and 1N is the same as z=1000. Obviously, if a number of people are concurrently running this agent in the same VW, then you should use distinct locations (and agent names). Additionally, try not to use location **(0, 0, 0)** or your avatar may appear inside citizens standing at ground zero.
- Set one `Sensor` element for each sensor required in your agent. For example,

```
<sensor class="vwadm.awa.base.AWAvatarSensor"/>
```

configures an avatar sensor,

```
<sensor class="vwadm.awa.base.AW3DObjectSensor">
    <parameter name="width" value="1000"/>
    <parameter name="height" value="1000"/>
</sensor>
```

configures a sensor that looks for 3D objects within a region **1000 × 1000** around the location of the MAS, and

```
<sensor class="vwadm.awa.base.AWChatSensor">
    <parameter name="recognisedCitizens" value="William"/>
</sensor>
```

configures a chat sensor that only recognises chat from citizen "William". Change "William" to your citizen name or do not enter a value at all.

- You can optionally configure an effector that the agent will use. For example,

```
<effector class="vwadm.awa.base.AWChatEffector"/>
```

It is optional because you can also create effectors as needed at runtime. For each configured effector, `ReteAgent` it asserts a definstance fact for that effector.

4 Using the AWAgent Configurator

The AWAgent Configurator is a utility that generates the `bat`, `properties`, `xml` configuration file and a template `clp` file for a single agent MAS. It is at a proof of concept stage only, *and it has only been tested on a single computer running Windows*. Current limitations are as follows:

- It currently only allows for a single agent MAS: you need to manually assemble separate XML files into a single XML file.
- Little or no validation of user input is done.
- The set of allowable domain names, ports, etc., are hard coded.
- Hideous user interface.

To run it, do the following:

- Download and unzip `base.zip` and `demos.zip` into a new directory on the machine you intend to run AWAgent on. Note that you can move to another directory later but you will need to edit the configuration files manually.
- Download http://mojo.csd.anglia.ac.uk/william/`AWbot/config/conf.jar` and save in the same directory.
- Double click on the configurator jar from the Windows File Explorer.
- Fill in the appropriate values in each field
- When finished, click on `Submit`. This will cause `bat`, `properties`, `xml`, and template `clp` files to be written into the current folder with filenames derived from the avatar name that you specified. This therefore means that the avatar name is mandatory.

Each time you click on the name of a Sensor, one sensor of that class will be configured and a corresponding window will appear. Each time you click on the `Add parameter` button a new parameter will be added into which you can set a new parameter. Similarly, each time you click on the name of an Effector, one effector of that class will be configured and a corresponding window will appear. Finally, one window appears with which agent parameters can be added.

You can add as many parameters as you want to the agent, sensors and effectors. Each will be available to the agent through the `getConfigurationParameter` method of `ReteAgent`. Each parameter is a single string that must be parsed by your agent. The only mandatory parameter is the agent parameter `"jess"` and is set automatically by the configurator.

If the following conditions all hold, the template `clp` file will contain a rule that recognises the 3D object from AW with which the agent represents itself:

1. There must be an agent parameter `"model"` containing the RWX model filename of the 3D object.
2. There must be an agent parameter `"description"` containing the description field of the 3D object.
3. There must be a sensor `vwadm.awa.base.AW3DObjectSensor`
4. There must be an effector `vwadm.awa.base.AW3DObjectEffector`

5 Sensors available in the software

Sense data pushed into Jess' working memory is given a fact name derived from its java class name and package name. For example, java class `Added3DObjectSenseData` in package `vwadm.awa.base` becomes fact `vwadm_awa_base_Added3DObjectSenseData`. All sense data are java beans that extend the java class `vwadm.awa.base.SenseData`. The following AW sensors are available in the main package `vwadm.awa.base`:

5.1 AWChatSensor

This receives chat from the VW. It provides `TextMessageSenseData` with the following Jess definstance slots:

- `text`, which is a String
- `sender`, which is a String
- `msgType`, which is an integer (0=said, 2=whisper)

The optional XML parameter `recognisedCitizens` restricts chat text to only those citizens. The parameter is a set of citizen names. The default is to recognise chat from all citizens. The parameter is intended for use where agents receive commands via chat or for efficiency reasons.

Java bean `TextMessageSenseData` extends `SenseData`.

5.2 AWAvatarSensor

This senses avatar adds, changes, deletions and clicks. For avatar delete it provides `AvatarSenseData` with the following Jess definstance slots:

- `name`, which is a String
- `session`, which is an integer
- `avatarDeleted`, which is TRUE

Java bean `AvatarSenseData` extends `SenseData`.

For avatar adds and changes it provides `LocatedAvatarSenseData` with the following Jess definstance slots:

- `name`, which is a String
- `session`, which is an integer
- `avatarDeleted`, which is FALSE
- `locn`, which is an object `Location6DF`, containing slots for `x`, `y`,`z`,`roll`,`yaw`,`tilt`.

Java bean `LocatedAvatarSenseData` extends `AvatarSenseData`.

5.3 AW3DSelectSensor

This senses when a citizen selects an object from the AW browser or when an agent generates an object select message. It provides `Selected3DSenseData` with the following Jess definstance slots:

- `avatarName`, which is a String
- `session`, which is an integer
- `objectNo`, which is an integer

Java bean `Selected3DSenseData` extends `SenseData`.

5.4 AW3DObjectSensor

This senses objects within a configured region. It looks for objects within the configured region about its initialised location. To change locations, shift the location of the agent (the `location` slot of the `vwadm_awa_base_ReteAgent` fact) as follows:

```
(?a setLocation ?alocn)
((?a getMAS) moveTo ?alocn)
```

where

`?alocn` is a `Location6DF`

`?a` is the value of the `OBJECT` slot of fact `vwadm_awa_base_ReteAgent`. The sensor accepts two configuration parameters: `height` and `width`. These are the `z` and `x` dimensions of a horizontal bounding box within which objects will be sensed.

For 3D object adds and changes, the sensor provides `AddedObject3DSenseData` with the following Jess definstance slots:

- `objectNo`, which is an integer

- `model`, which is a String
- `objectLocn`, which is an object `Location6DF`
- `ownerNo`, which is an integer
- `buildTimestamp`, which is an integer time in VRT (see VRTTimeSensor)
- `description`, which is a String
- `action`, which is a String
- `objectNo`, which is an integer

Java bean `AddedObject3DSenseData` extends `Object3DSenseData`, and java bean `Object3DSenseData` extends `SenseData`.

For object deletes it provides `DeletedObject3DSenseData` with the following Jess definstance slots:

- `objectNo`, which is an integer

Java bean `DeletedObject3DSenseData` extends `Object3DSenseData`.

For object clicks it provides `ClickedObject3DSenseData` with the following Jess definstance slots:

- `objectNo`, which is an integer
- `objectLocn`, which is an object `Location6DF`

Java bean `ClickedObject3DSenseData` extends `Object3DSenseData`.

For changes to an object by the 3D object effector (and only those changes), `ChangedObject3DSenseData` provides the following Jess definstance slots:

- `objectNo`, which is an integer
- `oldNo`, which is an integer

Java bean `ClickedObject3DSenseData` extends `Object3DSenseData`.

5.5 VRTTimeSensor

This extends `TimeSensor` to provide virtual reality (VR) time, with the following definstance slots:

- `period`, which is an integer number of seconds in VR time.

Java bean `TimeSenseData` extends `SenseData`.

With regard to time,

1. Time in the java class `java.util.Date` is the number of milliseconds since mid- night 1st Jan 1970 GMT
2. The Jess `(time)` function is the number of seconds since midnight 1st Jan 1970
3. The AW SDK only provides time in the universe attributes event, and that event only occurs at login time. The time provided is the number of seconds since midnight 1st Jan 1970 in the GMT time zone. Therefore, the agent gets this event, saves it, and thereafter use it to calculate the actual time. The AW browser, however, displays time as VRT (virtual reality time); VRT is the number of seconds since midnight 1st Jan 1970 in the GMT time zone -2hrs. Therefore, this sensor provides VRT time, not GMT, in order for the agents to use the same time as citizens see displayed on the browser.

The following shows how to extract calendar properties such as month from the binary VRT.

```
(defglobal ?*calendar* = (call Calendar getInstance))


(defrule time-rule-1
   ?f <- (MAIN::vwadm_awa_base_TimeSenseData (interval ?tm))
=>
   (retract ?f)
   (bind ?dt (call vwadm.awa.base.VRTTimeSensor vrtToDate ?tm))
   (call ?*calendar* setTime ?dt)
   (printout t "Day of week = "
               (?*calendar* get (get-member Calendar DAY_OF_WEEK)) crlf)
   (printout t "Month of year = "
               (?*calendar* get (get-member Calendar MONTH)) crlf)
```

5.6 Non-AW sensors

- `AgentChatSensor` to receive chat directly from another agent in the same MAS.
- `TimeSensor` to receive periodic time sense-data so as to drive polled tasks. It has one configuration parameter

  `period`, being the time period in seconds.

- `SQLSensor` to receive SQL results from a database via JDBC.
- `DOMSensor` to receive an arbitrary XML message from another agent.
- `ACLSensor` to send an ACL message to another agent.

6 Effectors available in the software

All effectors have the same architecture. Effectors can be created as needed - they do not need to be loaded at configuration time. However, if you do configure them, `ReteAgent` will maintain it as a dynamic java bean as a fact in working memory. To use an effector by dynamically creating at runtime, follow these steps:

1. Create and initialise an effector. For example, to create an AW chat effector from Jess,

```
(defglobal ?*awchat* = nil)
(defclass awchateffector AWChatEffector)
(bind ?*awchat* (new  AWChatEffector))
(?*awchat* init ?a nil))
```

2. Some effectors will have a `reset` method that should be used for each new effect-data.
3. For each effector property, there will be a set method. Again, for the AW chat effector from Jess,

```
(?*awchat* setMsg "Hi there! I like to look.")
(?*awchat* setMsgType (get-member MAS CHAT_SAID))
```

4. Once all of the properties are set, activate the effector.

```
(?*awchat* activate)
```

   The `activate` function will return an object. For many effectors, this object will be `nil`.

The method `setMsg` sets the `msg` property of the effector. Using a configured effector simplifies this by allowing you to use the Jess `(modify ...)` function on the effector's fact. In the following we will use this method. In order to use a configured effector, you must have used the Jess `(defclass ...)` on that effector before any rule requiring it fires. For example,

```
(defclass vwadm_awa_base_AWURLEffector AWURLEffector)
```

The following AW effectors are available in the main package `vwadm.awa.base`

6.1 AWChatEffector

This sends chat to AW, returning `nil`. The following properties exist:

- `msg`, which is a String
- `msgType`, which is an integer (0=said, 1=broadcast, 2=whisper)

An example rule that drives this effector is:

```
(defrule chat-action-rule
   (initialised)
   ?eff <- (vwadm_awa_base_AWChatEffector (OBJECT ?reference))
=>
   (modify ?eff (msg "Hi there!")
                (msgType (get-member MAS CHAT_SAID)))
   (call ?reference activate)
)
```

6.2 AWMoveEffector

This moves the avatar for the `MAS` to a new orientation by a defined distance, returning `nil`. However, the effector is best used indirectly via the `moveto` method of the `MAS` as follows:

```
(?a setLocation ?alocn)
((?a getMAS) moveTo ?alocn)
```

where
`?alocn` is a `Location6DF`
`?a` is the value of the `OBJECT` slot of fact `vwadm_awa_base_ReteAgent`.

6.3 AW3DObjectEffector

This adds, changes or deletes a 3D object to/in/from the world, returning an integer AWobject number. Note that if you change an existing 3D object it will be allocated a new number. The following properties exist:

- `command`, which is 0 for add, 1 for change, 2 for delete
- `x`, which is an integer
- `y`, which is an integer
- `z`, which is an integer
- `yaw`, which is an integer
- `roll`, which is an integer

- `tilt`, which is an integer
- `model`, which is a String
- `description`, which is a String
- `action`, which is a String
- `objNo`, which is an integer
- `oldx`, which is an integer that is only required CHANGE and only if driving the effector from java.
- `oldz`, which is an integer that is only required CHANGE and only if driving the effector from java.

An example rule that drives this effector to add an object is:

```
(defrule build-rule
   (build)
   ?eff <- (vwadm_awa_base_AW3DObjectEffector (OBJECT ?reference) (objNo 0))
=>
   (modify ?eff (command (get-member vwadm.awa.base.AW3DObjectEffector ADD))
                (x 100)
                (y 20)
                (z 100)
                (yaw 900)
                (roll 0)
                (tilt 0)
                (model "sign4")
                (description "Demo sign")
                (action "create sign \"A Sign\" color=silver bcolor=black;"))
   (call ?reference activate)
)
```

An example rule that drives this effector to change the just added object is:

```
(defrule move-rule
   ?f <- (move)
   ?eff <- (vwadm_awa_base_AW3DObjectEffector (OBJECT ?reference))
=>
   (modify ?eff (command (get-member vwadm.awa.base.AW3DObjectEffector CHANGE))
                (y 100))
   (call ?reference activate)
   (retract ?f)
   (assert (moved))
)
```

An example rule that drives this effector to delete the just changed object is:

```
(defrule demolish-rule
   ?f <- (demolish)
   ?eff <- (vwadm_awa_base_AW3DObjectEffector (OBJECT ?reference))
=>
   (modify ?eff (command (get-member vwadm.awa.base.AW3DObjectEffector DELETE)))
   (call ?reference activate)
   (retract ?f)
   (assert (demolished))
)
```

6.4 AWURLEffector

This changes the web page displayed on the right-hand side of a citizen's browser window. Beware that `AWURLEffector` will only work if you are a caretaker! You should also check whether the "Allow create url" feature is enabled for your VW (see the World Features dialog from the browser) The following properties exist:

- `url`, which is a String
- `session`, which is the integer session number for the citizen's browser

An example rule that drives this effector is:

```
(defrule avatar-sensedata-rule
   (initialised)
   ?eff <- (vwadm_awa_base_AWURLEffector (OBJECT ?reference))
   ?f <- (vwadm_awa_base_LocatedAvatarSenseData
             (name ?n)
             (avatarDeleted FALSE)
             (locn ?l)
             (session ?s))
=>
   (printout t "AVATAR " ?n " (" ?s ") IS AT " (?l toString) crlf)
   (modify ?eff (session ?s)
                (url " http://mojo.csd.anglia.ac.uk/william/AWbot/index.html"))
   (call ?reference activate)
   (retract ?f))
```

6.5 AWTeleportEffector

This teleports a citizen's avatar to another location in the VW or to another VW. Beware that this effector only works if the citizen that is running the agent has the EJECT right. The following properties exist:

- `world`, which is a String
- `session`, which is the integer session number for the citizen's browser
- `x`, which is an integer

- y, which is an integer
- z, which is an integer
- yaw, which is an integer
- warp, which is TRUE or FALSE.

6.6 EmailEffector and RestrictedEmailEffector

There are two versions of this effector. Both send an email message. EmailEffector will only work if you are citizen number 1; most users should run RestrictedEmailEffector. Both have the following properties:

- subject, which is a String
- filename, which is a String.

EmailEffector the following additional properties:

- fromCitizenName, which is a String
- toCitizenName, which is a String

RestrictedEmailEffector the following additional properties:

- fromCitizenEmail, which is a String
- toCitizenEmail, which is a String

Both require the following configuration parameters:

- SMTPPort
- mailhost
- domain

An example rule that drives EmailEffector effector is:

```
(defrule action-rule
   (initialised)
   ?eff <- (vwadm_awa_base_EmailEffector (OBJECT ?reference))
=>
   (modify ?eff (subject "Jess rules for emailtest agent")
                (filename "emailtest.clp")
                (fromCitizenName "William")
                (toCitizenName "William"))
   (call ?reference activate)
)
```

For RestrictedEmailEffector, replace fromCitizenName and toCitizenName with the email addresses of the sender and recipient respectively.

6.7 Non-AW effectors

- `DOMEffector` sends an arbitrary XML message to another agent or broadcasts to all agents in the MAS.
- `ACLEffector` to send an ACL message to another agent.

7 Perceptors available in the software

One combined sensor and perceptor is included in this software. It is `AWParsedChatSensor` and it extends chat text parsing to `AWChat-Sensor`. This receives chat from the VW, parses it, and provides either `ParsedChat` or `TextMessageSenseData`. The former is provided on errors; the normal case is that `ParsedChat` is provided with the following Jess definstance slots:

- `text`, which is a String
- `sender`, which is a String
- `msgType`, which is an integer (0=said, 2=whisper)
- `performative`, which is a String
- `prep`, which is a String
- `directobj`, which is a String
- `indirectobj`, which is a String

Java bean `ParsedChat` extends `TextMessageSenseData`, which in turn extends `SenseData`. Parsing is as follows. For whispered chat a single chat string is parsed as

```
verb [direct object] [preposition] [indirect object]
```

and normal chat is parsed as

```
[this agent's name,] verb [direct object] [preposition] [indirect object]
```

The VWADM AW Agent Package includes a small dictionary of verbs, prepositions and objects. Each of these can be overridden with user supplied dictionaries by writing an XML file and setting one of the following sensor parameters to a URL of that XML file.

- Parameter `verbsXMLurl` is the verb dictionary
- Parameter `prepsXMLurl` is the preps dictionary
- Parameter `objsXMLurl` is the object dictionary

An example of the verbs dictionary is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.0_01" class="java.beans.XMLDecoder">
 <object class="java.util.HashMap">
  <void method="put">
   <string>^\s*plan</string>
   <string>ORGANISE</string>
  </void>
  <void method="put">
   <string>^\s*search</string>
   <string>SELECT</string>
  </void>
 </object>
</java>
```

Each `void` element declares a new verb. The first `string` element is a java regular expression to match to the chat string (see the Java package `java.util.regexp`), and the second is a symbol to parse the matched verb as.

The parser looks for the verb first. If found, it then looks for a preposition. Anything between the verb and the preposition is the direct object; anything after the preposition is the indirect object.

8 Procedure to run the office demo

It is recommended that you try the simple test agents from `demos.zip` first. These are those corresponding to the simple rules in Section 6.

The office demo as described in this guide contains only a simple intelligent door. This demo will not work unless you own the door object. To try the demo you need to create a door that your agent will have permission to change. If that is the case then put all of the files listed in Section 2 into one directory. The supplied files assume that this directory is called `\demos`.

Make the changes described in Section 3, and open an MS-DOS command shell (this is strongly recommended because it prints messages that help with debugging). Change to that directory and type

```
newoffice
```

This will run the newoffice.bat file. If you have created your own agent and have another bat file, the general procedure is to type the name of the bat file.

Jess rules that use `AWChatEffector` will print messages to the AW browser chat window. Jess rules that use the Jess function `(printout)` or print to the java standard output stream will print to the MS-DOS command shell window. By moving your citizen's avatar around the VW, you can find locations with greater resolution that is shown on the AW browser using the printout effector to print the location of the avatar.

To end the session you can either enter control-C in the MS-DOS command shell, or alternatively, chat sensor rules could look for a chat message from a recognised citizen that contains

```
die die die
```

When you type this in the AW chat window, the message would be detected by the agent and the MAS dies.

9 Demonstration

This section describes a demonstration MAS named "Office" that contains an agent named "door".

9.1 Configuration for `Office`

office.xml

```
1  <?xml version="1.0" ?>
2  <!DOCTYPE MAS SYSTEM " http://mojo.csd.anglia.ac.uk//william/MAS.dtd">
3  <MAS>
4  <connection class="vwadm.awa.base.AWConnection">
5     <parameter name="domain" value=" http://mojo.csd.anglia.ac.uk/"/>
6     <parameter name="port" value="5670"/>
7     <parameter name="avatarType" value="22"/>
8     <parameter name="avatarname" value="Office"/>
9     <parameter name="worldname" value="Agents"/>
10 </connection>
11
12 <reteagent name="door">
13 <location x="-815" y="-250" z="13848"/>
14 <parameter name="jess"
15    value=" http://mojo.csd.anglia.ac.uk/william/AWbot/demos/newdoor.clp"/>
16 <parameter name="owner" value="William"/>
17 <parameter name="friendlist" value="William Austin"/>
18 <parameter name="officeXmin" value="-1020"/>
19 <parameter name="officeXmax" value="635"/>
20 <parameter name="officeZmin" value="12120"/>
21 <parameter name="officeZmax" value="13979"/>
22 <parameter name="radius" value="300"/>
23 <behaviours codebase="file:">
24 <sensor class="vwadm.awa.base.AW3DObjectSensor">
25    <parameter name="width" value="500"/>
26    <parameter name="height" value="500"/>
27 </sensor>
28 <sensor class="vwadm.awa.base.AWAvatarSensor"/>
29 <sensor class="vwadm.awa.base.AWChatSensor"/>
30 <effector class="vwadm.awa.base.AW3DObjectEffector"/>
31 <effector class="vwadm.awa.base.AWChatEffector"/>
32 </behaviours>
```

```
33  </reteagent>
34
35  </MAS>
```

XML element `MAS` configures the MAS to run in the VW with name "Agents" and avatar number 22. The remainder of the configuration describes a `reteagent` element for the agent. A MAS can contain any number of `reteagent` elements. Each `reteagent` element contains a number of `parameter` elements, one `location` element, and one `behaviour` element. Any number of `parameter` elements, with arbitrary names and values, can be included to configure your agent but the parameter `jess` is mandatory. For "door", the following parameters are read by `newdoor.clp`:

- `owner` is the name of the citizen that is to own the door.
- `friendlist` is a space-delimited list of friends of the door. It should contain at least the owner.
- `officeXmin` is the minimum `x` value of an X-Z axis aligned rectangular bounding box that `door.clp` deems to be the office.
- `officeXmax` is the maximum `x` value of an X-Z axis aligned rectangular bounding box that `door.clp` deems to be the office.
- `officeZmin` is the minimum `z` value of an X-Z axis aligned rectangular bounding box that `door.clp` deems to be the office.
- `officeZmax` is the maximum `x` value of an X-Z axis aligned rectangular bounding box that `door.clp` deems to be the office.
- `radius` is the radius of a vertical cylindrical space about the agent's location that the door considers to be the doorway for the purposes of classifying avatar locations.

The horizontal region of the VW that is occupied by the office is configured by the above globals. A better alternative would be to use a java.awt.Shape. The agent would sense wall objects, compute a 2D projection of the affine transformed walls, and compute as a java.awt.Polygon the convex hull of the vertices of the transformed and projected walls.

This particular demonstration applies agency to a pre-existing 3D door object. Therefore this object must already exist in the VW and be owned by the citizen configured in `newoffice.properties`. The following object should exist:

- A door with model `pp16w3.rwx`, description "Intelligent Door", location `x="-815" y="-250" z="13848"`, and the following action:

```
"create name d1, color white, animate me zspace 1 1 200, astop;
adone move 0 4 0 noloop time=1 wait=2"
```

To run the demo, type `newoffice` in the MS-DOS command window to execute the newoffice.bat file. Either

- Type `<control-C>` from the MS-DOS command window, or
- Have the owner of the recorder type `die die die`

to kill the agent.

9.2 Annotated Jess fact base for agent `door`

The sensors are written in java, so the first thing to do is to specify where to find the relevant java classes for the sensor data.

```
(import jess.*)
(import vwadm.awa.base.*)
```

The sensors use java Beans to store the sense data. Jess provides the `defclass` construct to automatically generate a template that represents a class of java Beans. Each java bean property is seen by Jess rules as a slot in an unordered fact. In addition, three slots are always defined:

- slot `OBJECT` is a reference to the java bean instance
- slot `class` is the object returned by the `getClass` method that every java instance has
- slot `name` is the name of the bean.

`ReteAgent` takes advantage of `defclass` by pushing `SenseData`, as well as a reference to the `vwadm.awa.base.ReteAgent` instance, into Jess working memory as java beans. `ReteAgent` only pushes beans into working memory for which `defclass` templates exist. You should remember that sensors keep pushing sense data into working memory - you need to ensure that your rules eventually retract that sense data.

This means you need a `defclass` for those beans, that is, for the sense data facts, that are required for your rules. Below is the list of `defclass` statements for the sense data used by `door`. In the `defclass` statement, the name of the fact in Jess is first (the long name) and the name of the java Bean is second. In your rules, you will use the long name. The slots for these facts are defined in this guide in the section 5.

```
(clear)

(defclass vwadm_awa_base_Agent Agent)
(defclass vwadm_awa_base_ReteAgent ReteAgent extends vwadm_awa_base_Agent)
(defclass vwadm_awa_base_AW3DObjectEffector AW3DObjectEffector)
```

```
6   (defclass vwadm_awa_base_AWChatEffector AWChatEffector)
7
8   (defclass vwadm_awa_base_SenseData SenseData)
9   (defclass vwadm_awa_base_Object3DSenseData Object3DSenseData
10     extends vwadm_awa_base_SenseData)
11  (defclass vwadm_awa_base_AddedObject3DSenseData AddedObject3DSenseData
12     extends vwadm_awa_base_Object3DSenseData)
13  (defclass vwadm_awa_base_DeletedObject3DSenseData DeletedObject3DSenseData
14     extends vwadm_awa_base_Object3DSenseData)
15  (defclass vwadm_awa_base_AvatarSenseData AvatarSenseData
16     extends vwadm_awa_base_SenseData)
17  (defclass vwadm_awa_base_LocatedAvatarSenseData  LocatedAvatarSenseData
18     extends vwadm_awa_base_AvatarSenseData)
19  (defclass vwadm_awa_base_TextMessageSenseData TextMessageSenseData
20     extends vwadm_awa_base_SenseData)
```

The `extends` construct tells Jess that one bean is a specialisation of another, allowing for inheritance of properties. You can also use `extends` on deftemplate facts. If you want to know what the slots of a defclass fact are without looking through java code, do something like this:

```
(printout t (ppdeftemplate vwadm_awa_base_ReteAgent) crlf)
```

To make the agent model clear, the implementation has rules and facts in separate modules. The following modules are for rules:

- INTERPRETATION
- HYPOTHESISER
- ACTION

A few additional rules run in module MAIN. This is because that is the default module and so it is there where the scheduler rules should go. These four modules represent the process model of the agent.

The following modules are for facts:

- MAIN
- STRUCTURE
- BEHAVIOUR
- FUNCTION

As sensors and effectors use the default module, sense data use the MAIN module. Facts about the agent, such as the schedule below, are also in the MAIN module.

The following `deftemplate` statements define the names and slots of the unordered facts in the `door` MAIN module. MAIN holds sense data, effectors, and other facts that are about the agent but are not FBS representations.

```
(deftemplate MAIN::schedule
   (slot collection (type OBJECT)))

(deftemplate MAIN::goal
   (slot label (type LEXEME) (default NONE))
   (multislot params (default (create$))))
```

The following `deftemplate` statements define the names and slots of the unordered facts in the `door` STRUCTURE module.

```
;Structure of a door object.
(deftemplate STRUCTURE::door
   (slot object-id (type INTEGER) (default 0))    ;identifier of door
   (slot location (type OBJECT) (default  nil))   ;location of door
   (slot ownerid (type INTEGER) (default  0))     ;owner id
   (slot owner  (type STRING))                    ;owner name
   (slot radius (type INTEGER) (default 200))     ;recognition radius
   (slot model (type STRING) (default "pp16w3.rwx"))
   (slot description(type STRING) (default "Intelligent Door"))
   (slot action (type STRING) (default "create color white"))
)

;Structure of a person in the world
(deftemplate STRUCTURE::person
   (slot name (type STRING))          ;name of person
   (slot location (type OBJECT))      ;the location of the person
   (slot session (type INTEGER))
   (slot chat (type STRING))          ;What person said
)
```

The following `deftemplate` statements define the names and slots of the unordered facts in the `door` BEHAVIOUR module.

```
12      (slot name (type STRING))
13      (slot area (type LEXEME) (default NONE))
14      (slot state (type LEXEME) (default NONE)))
15
16   (deftemplate BEHAVIOUR::security
17      (slot name)
18      (multislot chat))
```

`BEHAVIOUR::door` is state based:

- door state is one of *NONE, OPEN, CLOSED*
- person state is one of *NONE, SECURITY PENDING, FRIEND*
- person area is one of *NONE, AWAY, DOORWAY, OFFICE*

The following `deftemplate` statements define the names and slots of the unordered facts in the `door` FUNCTION module.

```
1    (deftemplate FUNCTION::door
2       (slot object-id (type INTEGER) (default 0))
3       (slot selected)) ;the currently selected function
```

The following statements define global variables and initial facts.

```
1    (defglobal ?*officeXmin* = nil)
2    (defglobal ?*officeXmax* = nil)
3    (defglobal ?*officeZmin* = nil)
4    (defglobal ?*officeZmax* = nil)
5
6    ;This holds a reference to the object effector. It is only used by the
7    ;terminator function.
8    (defglobal ?*objeff* = nil)
9
10   (deffacts MAIN::initial-FBS
11      (MAIN::goal)
12      (STRUCTURE::door)
13      (BEHAVIOUR::door)
14      (FUNCTION::door)
15   )
16   (reset)
```

9.3 Annotated Jess rules for agent `door`

This is only a demonstration agent. There are lots of things that it could do but doesn't, or that it could do better, or just do differently. It also undoubtedly still has bugs.

Below you will find the following in the start-up rule:

```
(modify ?a (traceEnabled TRUE))
```

This enables a debugging trace. Comment out this line to disable it. Beaware that the trace prints out heaps of stuff that is invaluable for debugging.

9.3.1 MAIN module

The MAIN module has on start-up rule.

```
1  (defrule MAIN::start-up-rule
2     (declare (salience 10))
3     ?a <- (vwadm_awa_base_ReteAgent (OBJECT ?ag))
4  =>
5     (set-multithreaded-io TRUE)
6
7     ;(printout t (ppdeftemplate vwadm_awa_base_ReteAgent) crlf)
8     (modify ?a (traceEnabled TRUE))
9
10    ;We use java.lang.Integer.parseInt here because the defglobals are
11    ;untyped strings
12    (bind ?*officeXmin* (call java.lang.Integer
13                              parseInt
14                              (?ag getConfigurationParameter "officeXmin")))
15    (bind ?*officeXmax* (call java.lang.Integer
16                              parseInt
17                              (?ag getConfigurationParameter "officeXmax")))
18    (bind ?*officeZmin* (call java.lang.Integer
19                              parseInt
20                              (?ag getConfigurationParameter "officeZmin")))
21    (bind ?*officeZmax* (call java.lang.Integer
22                              parseInt
23                              (?ag getConfigurationParameter "officeZmax")))
24
25    (assert (MAIN::schedule (collection
26                  (new java.util.Vector
27                     (call java.util.Arrays
28                        asList
29                        (create$ ACTION HYPOTHESISER INTERPRETATION MAIN))))
30    ))
31 )
```

The MAIN module also has a scheduler. The scheduler requires that you use `extends vwadm_awa_base_SenseData` on each sense data defclass or else it won't trigger the scheduler rule when new sense data arrives.

We implement the scheduler with the Jess focus stack, restricting rules to firing within one module at a time. The reason is to ensure that the control mechanism is isolated from any knowledge: all rules on the agenda from the current focus module run, then all rules on the agenda from the next scheduled module run, and so on until we get back to the MAIN module (which will then trigger off the next sense data). The fact `(schedule (collection ?s))` is a collection of module names to schedule **in reverse order**. Make ?s an instance of a java collection class that provides an iterator method.

The next rule controls scheduling. Rules could, if desired, alter `(schedule ...)` to change which set of intentions are current.

```
(deffunction scheduler-iterate (?s)
   (bind ?it (?s iterator))
   (while (?it hasNext)
        (bind ?m (?it next))
        (focus ?m)))

;Schedule all modules everytime something changes
(defrule MAIN::scheduler-rule
   (declare (salience 5))
   (MAIN::initialised)
   (vwadm_awa_base_SenseData)
   (MAIN::schedule (collection ?s))
=>
   (scheduler-iterate ?s))
```

The final rule in MAIN is a termination cleanup rule. ReteAgent sets a default Jess function `MAIN::terminate` that is called before the agent dies. Therefore, you can override this function with your own if the agent needs to clean up after itself before it dies. Be careful, however, as this needs to be threadsafe. Read the javadoc documentation on the `addShutdownHook` method in the java class `java.lang.Runtime` for details on what you shouldn't do. In this demo the termination function restores the default action to the door so that it will still open when the agent is not running. Note that the string should be on one line, not split as is printed below.

```
(deffunction MAIN::terminate ()
   (if (neq ?*objeff* nil) then
      (call ?*objeff* setAction "create name d1, color white, animate me
zspace 1 1 200, astop; adone move 0 4 0 noloop time=1 wait=2")
      (call ?*objeff* activate)))
```

9.3.2 INTERPRETATION module

This first INTERPRETATION rule assumes that the only 3D object "pp16*" within the configured region that has the description `(door ... (description ?d))` is the one that is to be intelligent. A more reliable alternative is to have this agent build the door itself, but in that case the door disappears when the agent is not running. The rule takes data from the configuration and puts it in the fact base for the door that matches the LHS sense data. It also initialises the effector for the door.

```
(defrule INTERPRETATION::initialise-door
   "Find an intelligent door"
   ?f <- (MAIN::vwadm_awa_base_AddedObject3DSenseData
                                          (objectNo ?no)
                                          (model ?m)
                                          (description ?desc)
                                          (objectLocn ?locn)
                                          (ownerNo ?own))
   ?ds <- (STRUCTURE::door (object-id 0)
                           (model ?m)
                           (action ?act)
                           (description ?desc))
   ?db <- (BEHAVIOUR::door)
   ?df <- (FUNCTION::door)
   ?eff <- (MAIN::vwadm_awa_base_AW3DObjectEffector (OBJECT ?o))
   ?a <- (MAIN::vwadm_awa_base_ReteAgent (OBJECT ?ag))
=>
   (bind ?*objeff* ?o)
   (bind ?radius (call java.lang.Integer
                       parseInt
                       (?ag getConfigurationParameter "radius")))
   (bind ?own (?ag getConfigurationParameter "owner"))
   (bind ?flist (?ag getConfigurationParameter "friendlist"))
   (modify ?ds (object-id ?no)
               (location ?locn)
               (ownerid ?own)
               (owner ?own)
               (radius ?radius))
   (printout t "DOOR IS AT " (?locn toString) crlf)

   (modify ?db (object-id ?no) (friendlist (explode$ ?flist)))
   (modify ?df (object-id ?no) (selected RESTRICT-ACCESS))

   (bind ?x (get-member ?locn x))
   (bind ?z (get-member ?locn z))
   (modify ?eff (objNo ?no)
                (action ?act)
                (description ?desc)
                (model ?m)
                (x ?x)
                (oldx ?x)
```

```
                    (y (get-member ?locn y))
                    (z ?z)
                    (oldz ?z)
                    (yaw (get-member ?locn yaw))
                    (roll (get-member ?locn roll))
                    (tilt (get-member ?locn tilt)))
    (retract ?f)
)
```

The door does not need to know about other 3D objects in the VW so that sense data is removed:

```
(defrule INTERPRETATION::discard-other-objects
   ?f <- (MAIN::vwadm_awa_base_AddedObject3DSenseData
                                          (objectNo ?no)
                                          (model ?m)
                                          (description ?d)
                                          (objectLocn ?locn)
                                          (ownerNo ?own))
   (STRUCTURE::door (model ?m2) (description ?d2))
   (test (or (neq ?m2 ?m) (neq ?d2 ?d)))
=>
   (retract ?f)
)
```

Create a STRUCTURE and BEHAVIOUR for a located person that was previously unknown to the fact base:

```
(defrule INTERPRETATION::person-INTERPRETATION-1
   ?f <- (MAIN::vwadm_awa_base_LocatedAvatarSenseData
             (name ?n)
             (locn ?l)
             (session ?s))
   (not (STRUCTURE::person (name ?n)))
=>
   (assert (STRUCTURE::person (location ?l) (session ?s) (name ?n)))
   (assert (BEHAVIOUR::person (name ?n)))
   (retract ?f)
)
```

Update STRUCTURE and BEHAVIOUR for a located person that is already known to the fact base:

```
(defrule INTERPRETATION::person-INTERPRETATION-2
   "Locate a known person"
   ?f <- (MAIN::vwadm_awa_base_LocatedAvatarSenseData
            (name ?n)
            (locn ?l)
            (session ?s))
   ?ps <- (STRUCTURE::person (name ?n))
   ?pb <- (BEHAVIOUR::person (name ?n))
=>
   (modify ?ps (location ?l))
   (modify ?pb (area NONE))
   (retract ?f)
)
```

Remove a deleted person from the fact base:

```
(defrule INTERPRETATION::person-INTERPRETATION-3
   ?f <- (MAIN::vwadm_awa_base_AvatarSenseData (name ?n) (avatarDeleted TRUE))
   ?ps <- (STRUCTURE::person (name ?n))
   ?pb <- (BEHAVIOUR::person (name ?n))
=>
   (retract ?f ?ps ?pb)
)
```

This detects a message by a non-friend. After being asked
"Hi there ¡visitor-name¿ who do you wish to visit?"
by another rule, the visitor must answer with a name (and only the name) that is on the friend list. If they do, they receive a security clearance. We could make this agent smarter by giving this person a temporary clearance instead of adding them to the friend list.

```
(defrule INTERPRETATION::text-INTERPRETATION-1
   ?f <- (MAIN::vwadm_awa_base_TextMessageSenseData (text ?t)
                                                    (sender ?visitor))
   (BEHAVIOUR::person (name ?visitor) (area DOORWAY) (state SECURITYPENDING))
   (BEHAVIOUR::door (friendlist $? ?friend $?))
   (test (?t equals ?friend))
=>
   (assert (BEHAVIOUR::security (name ?visitor)
                                (chat (explode$ ?t))))
   (retract ?f)
)
```

This is a clean way to kill agent: owner must say “die die die”:

```
(defrule INTERPRETATION::death-rule
   ?f <- (MAIN::vwadm_awa_base_TextMessageSenseData (text ?t)
                                                    (sender ?own))
   (STRUCTURE::door (owner ?own))
   (test (neq (str-index "die die die" ?t) FALSE))
=>
   (exit)
)
```

This throws away chat text that is not interpreted by other rules. The rule is of low salience: it acts as a default rule:

```
(defrule INTERPRETATION::text-INTERPRETATION-2
   (declare (salience -10))
   ?f <- (MAIN::vwadm_awa_base_TextMessageSenseData (text ?t))
=>
   (retract ?f)
)
```

This function tests whether a person is on the friend list:

```
(deffunction INTERPRETATION::recognised (?sender $?list)
   (foreach ?s $?list (if (?sender equals ?s) then (return TRUE)))
   (return FALSE)
)
```

Any unclassified person that is within ?rad of the vertical axis of the door is classified as being in the doorway:

```
(defrule INTERPRETATION::classify-person-1
   (declare (salience 10))
   ?pb <- (BEHAVIOUR::person (name ?n) (area NONE))
   ?ps <- (STRUCTURE::person (name ?n) (location ?pl))
   (STRUCTURE::door (radius ?rad)
                (location ?dl&:(neq ?dl nil)&:(< (?pl distance ?dl) ?rad)))
=>
   (modify ?pb (area DOORWAY))
)
```

Any unclassified person that is within the office 2D bounds is classified as in the office. Note that this rule has lower salience than the doorway rule above.

```
(defrule INTERPRETATION::classify-person-2
   (declare (salience 5))
   ?pb <- (BEHAVIOUR::person (name ?n) (area NONE))
   ?ps <- (STRUCTURE::person (name ?n) (location ?pl))
   (STRUCTURE::door (radius ?rad) (location ?dl&:(neq ?dl nil)))
   (test (?pl inBounds ?*officeXmin*
                       ?*officeXmax*
                       ?*officeZmin*
                       ?*officeZmax*))
=>
   (modify ?pb (area OFFICE))
)
```

Any unclassified person not in the doorway or office is classified as being away:

```
(defrule INTERPRETATION::classify-person-3
   (declare (salience 5))
   ?pb <- (BEHAVIOUR::person (name ?n) (area NONE))
   ?ps <- (STRUCTURE::person (name ?n) (location ?pl))
   (STRUCTURE::door (radius ?rad) (location ?dl&:(neq ?dl nil)))
   (test (>= (?pl distance ?dl) ?rad))
   (test (not (?pl inBounds ?*officeXmin*
                            ?*officeXmax*
                            ?*officeZmin*
                            ?*officeZmax*)))
=>
   (modify ?pb (area AWAY))
)
```

Anyone not known as a friend that is on the friend list is denoted a friend:

```
(defrule INTERPRETATION::classify-person-4
   (declare (salience 5))
   ?pb <- (BEHAVIOUR::person (name ?n) (state ~FRIEND))
   ?ps <- (STRUCTURE::person (name ?n) (location ?pl))
   (BEHAVIOUR::door (object-id ?id&:(neq ?id 0)) (friendlist $?list))
   (test (INTERPRETATION::recognised ?n $?list))
=>
   (modify ?pb (state FRIEND))
)
```

Any non-friend that is in the doorway should be asked who they want to visit:

```
(defrule INTERPRETATION::classify-person-5
   (declare (salience 5))
   ?pb <- (BEHAVIOUR::person (name ?n) (state NONE) (area DOORWAY))
   ?ps <- (STRUCTURE::person (name ?n) (location ?pl) (session ?s))
   ?d <- (BEHAVIOUR::door (object-id ?id&:(neq ?id 0)) (friendlist $?list))
   (test (not (INTERPRETATION::recognised ?n $?list)))
=>
   (assert (MAIN::goal (label CHAT)
                       (params ?s (str-cat "Hi there "
                                           ?n
                                           ", who do you wish to visit?"))) )
   (modify ?pb (state SECURITYPENDING))
)
```

Any non-friend that is in the doorway and has a security clearance should be changed to be a friend:

```
(defrule INTERPRETATION::classify-person-6
   (declare (salience 5))
   ?pb <- (BEHAVIOUR::person (name ?n) (state SECURITYPENDING) (area DOORWAY))
   ?ps <- (STRUCTURE::person (name ?n) (location ?pl))
   ?f <- (BEHAVIOUR::security (name ?n) (chat $? ?friend $?))
   (BEHAVIOUR::door (object-id ?id&:(neq ?id 0)) (friendlist $? ?friend $?))
=>
   (modify ?pb (state FRIEND))
   (retract ?f)
)
```

And finally, this is a default security rule.

```
(defrule INTERPRETATION::classify-person-7
   (declare (salience -10))
   ?f <- (BEHAVIOUR::security (chat $?))
=>
   (retract ?f)
)
```

9.3.3 Hypothesiser modules

Select function ALLOW-ACCESS whenever there is a friend in the doorway and there is not also a non-friend:

```
(defrule HYPOTHESISER::evaluate-rule-1
   ?f <- (FUNCTION::door (selected ~ALLOW-ACCESS)
                         (object-id ?id&:(neq ?id 0)))
   (BEHAVIOUR::person (area DOORWAY) (state FRIEND))
   (not (BEHAVIOUR::person (area DOORWAY) (state ~FRIEND)))
=>
   (modify ?f (selected ALLOW-ACCESS)))
```

Select function RESTRICT-ACCESS whenever there is a non-friend in the doorway:

```
(defrule HYPOTHESISER::evaluate-rule-2
   ?f <- (FUNCTION::door (selected ~RESTRICT-ACCESS)
                         (object-id ?id&:(neq ?id 0)))
   (BEHAVIOUR::person (area DOORWAY) (state ~FRIEND))
=>
   (modify ?f (selected RESTRICT-ACCESS)))
```

Select function RESTRICT-ACCESS whenever there is not a friend in the doorway:

```
(defrule HYPOTHESISER::evaluate-rule-3
   ?f <- (FUNCTION::door (selected ~RESTRICT-ACCESS)
                         (object-id ?id&:(neq ?id 0)))
   (not (BEHAVIOUR::person (area DOORWAY)))
=>
   (modify ?f (selected RESTRICT-ACCESS)))
```

If the selected function is to restrict access and we don't currently have such a goal, then assert one:

```
(defrule HYPOTHESISER::synthesise-rule-1
   (FUNCTION::door (selected RESTRICT-ACCESS)
                   (object-id ?id&:(neq ?id 0)))
   ?f <- (BEHAVIOUR::door (state ~CLOSED))
   (not (MAIN::goal (label CLOSE)))
=>
   (assert (MAIN::goal (label CLOSE))))
```

If the selected function is to allow access and we don't currently have such a goal, then assert one:

```
(defrule HYPOTHESISER::synthesise-rule-2
   (FUNCTION::door (selected ALLOW-ACCESS)
                   (object-id ?id&:(neq ?id 0)))
   ?f <- (BEHAVIOUR::door (state ~OPEN))
   (not (MAIN::goal (label OPEN)))
=>
   (assert (MAIN::goal (label OPEN))))
```

9.3.4 Action modules

Satisfy goal OPEN door by changing the door in the VW such that it is open:

```
(defrule ACTION::open-door-1
   (MAIN::vwadm_awa_base_ReteAgent (OBJECT ?a))
   ?d <- (BEHAVIOUR::door (object-id ?id&:(neq ?id 0)) (state ~OPEN))
   ?ds <- (STRUCTURE::door (object-id ?id))
   ?df <- (FUNCTION::door (object-id ?id))
   ?f <- (MAIN::goal (label OPEN))
   ?eff <- (MAIN::vwadm_awa_base_AW3DObjectEffector (OBJECT ?reference)
                                                    (objNo ?id&:(neq ?id 0)))
=>
   (modify ?eff (command (get-member vwadm.awa.base.AW3DObjectEffector CHANGE))
                (action "create solid off, visible off"))
   (bind ?no (?reference activate))
   (retract ?f)
   (modify ?d (state OPEN) (object-id (?reference getObjNo)))
   (modify ?ds (object-id (?reference getObjNo)))
   (modify ?df (object-id (?reference getObjNo)))
)
```

We satisfy goal OPEN door when the door is already open by doing nothing:

```
(defrule ACTION::open-door-2
   (BEHAVIOUR::door (state OPEN))
   ?f <- (MAIN::goal (label OPEN))
=>
   (retract ?f)
)
```

Satisfy goal CLOSE door by changing the door in the VW such that it is closed:

```
(defrule ACTION::close-door-1
   (MAIN::vwadm_awa_base_ReteAgent (OBJECT ?a))
   ?d <- (BEHAVIOUR::door (object-id ?idi&:(neq ?id 0)) (state CLOSED))
   ?ds <- (STRUCTURE::door (object-id ?id))
   ?df <- (FUNCTION::door (object-id ?id))
   ?f <- (MAIN::goal (label CLOSE))
   ?eff <- (MAIN::vwadm_awa_base_AW3DObjectEffector (OBJECT ?reference)
                                                    (objNo ?id&:(neq ?id 0)))
=>
   (modify ?eff (command (get-member vwadm.awa.base.AW3DObjectEffector CHANGE))
                (action "create color white"))
   (bind ?no (?reference activate))
   (modify ?d (state CLOSED) (object-id (?reference getObjNo)))
   (modify ?ds (object-id (?reference getObjNo)))
   (modify ?df (object-id (?reference getObjNo)))
   (retract ?f)
)
```

We satisfy goal CLOSED door when the door is already closed by doing nothing:

```
(defrule ACTION::close-door-2
   (BEHAVIOUR::door (state CLOSED))
   ?f <- (MAIN::goal (label CLOSE))
=>
   (retract ?f)
)
```

Satisfy goal CHAT with params ?s and ?msg (being a citizen session number and chat message, respectively) by whispering a chat message to a citizen:

```
(defrule ACTION::chat-1
   ?f <- (MAIN::goal (label CHAT) (params ?s ?msg))
   ?eff <- (MAIN::vwadm_awa_base_AWChatEffector (OBJECT ?reference))
=>
   (modify ?eff (msg ?msg)
                (targetSession ?s)
                (msgType (get-member MAS CHAT_WHISPER)))
   (call ?reference activate)
   (retract ?f)
)
```

9.4 Using agent `door`

The office for person area purposes is just a rectangular region. You configure this region in the XML. If a friend walks up and stops in the DOORWAY, the door will open. If anyone else does, it will send a chat message asking who they want to visit. The visitor reponds

```
William
```

for example. If the name is of a person that is currently in the office, then that person becomes a friend.

Appendix D. Advanced User Guide for the VWADM's AW Agent Package

Advanced User Guide for
the Virtual World Application Design Method's
Active Worlds Agent Package

William Sawyerr
Department of Computing and Technology
Anglia Ruskin University

Contents

1 Communication between agents

Fundamental to multi-agent systems is communication that allows agents to enlist the support of other agents to achieve goals, commit to the execution of actions, report on progress, and so on (Cohen & Levesque, 1995). This section describes how to use an ACL effector and sensor to send messages directly between agents, which means without going through the AW server. This allows agents to cooperate and negotiate without flooding a VW with chat that is meaningless to citizens.

The conceptual basis for agent communication languages (ACLs) is the theory of speech acts (Austin, 1962; Searle, 1969; Cohen & Perrault, 1979). In AWAgent, the FIPA ACL is encapsulated within `ACLSensor` and `ACLEffector`. These adopt the FIPA ACL abstract message structure but use a simple XML implementation that is transparent to users.

Austin (1962) describes three aspects of a speech act: locution, illocution and perlocution. Locution is the physical utterance, which for the agents is the actual message sent by the effector. Illocution is the meaning of the locution; it is the receiver's belief of what the sender intended by the locution. Perlocution is the action that results from the illocution. Performatives are an illocutionary force interpreted from the locution. The FIPA ACL describes a set of 22 performatives that constitute the communicative acts (FIPA, 2002). The most important FIPA performatives are `INFORM` and `REQUEST`. FIPA (2002) defines an abstract message structure; each message can contain the following elements:

FIPA Element	FIPA Mandatory?	`ACLSensor` and `ACLEffector`
performative	yes	mandatory
sender	no	automatic
receiver	no	automatic
reply-to	no	optional
content	yes	mandatory
language	no	optional
encoding	no	optional
ontology	no	optional
protocol	no	optional
conversation-id	no	automatic
reply-with	no	optional
in-reply-to	no	optional
reply-by	no	optional

`ACLEffector` sends an ACL message to either another agent or as a broadcast to all agents in the `MAS`. It uses a message server and a thread pool to deliver messages asynchronously without going via the AW server. The following properties exist:

. `target`, which is an object that should be a java reference to the MAS or an agent. To get a reference to an agent, given that `?name` is a Jess variable holding a string that is the name of the intended receiving agent and `?ag` is the value of the `OBJECT` slot of the `vwadm_awa_base_ReteAgent` fact:

```
1  (bind ?targ ((?ag getMAS) findAgent ?name))
```

To get a reference to the `MAS` for a broadcast:

```
1  (bind ?targ (?ag getMAS))
```

. `performative`, which is an integer. Java class `ACLEffector` defines the allowable values.

. `content`, which is an array (a Jess list) of XML-serializable Java `Objects`. AWAgent provides example content classes that can be used as ordered facts for this purpose or extended as desired:

  - `vwadm.awa.base.Content`, which is an abstract base class that should not be instantiated.
  - `vwadm.awa.base.Action`, which extends `vwadm.awa.base.Content` and provides the following slots:
    * `actor`, which is a string.
    * `act`, which is a string.
    * `arguments`, which is an array (a Jess list) of strings.
  - `vwadm.awa.base.Proposition`, which extends `vwadm.awa.base.Content` and provides the following slots:
    * `predicate`, which is a string.
    * `obj`, which is a string.
    * `subject`, which is a string.
    * `belief`, which is boolean.
  - `vwadm.awa.base.StringContent`, which extends `vwadm.awa.base.Content` and provides the following slot:
    * `str`, which is a string.

· `language`, which is a string.

· `encoding`, which is a string.

· `protocol`, which is a string.

· `conversationid`, which is a string but may change to be an integer.

· `replywith`, which is a string.

· `inreplyto`, which is a string but may change to be an integer.

· `replyby`, which is a string but may change to be an integer.

`ACLSensor` receives an ACL message and provides `ACLSenseData` Java beans to the ReteAgent working memory. `ACLSenseData` has the same Jess definstance slots as the ACLEffector.

1.1 Demonstration sender and receiver agents

1.1.1 Configuration

txrxtest.xml

```
<?xml version="1.0" ?>
<!DOCTYPE MAS SYSTEM "http://mojo.csd.anglia.ac.uk/william/MAS.dtd">
<MAS>

<connection class="vwadm.awa.base.Connection">
   <parameter name="avatarname" value="txrxtest"/>
</connection>

<reteagent name="rxtest">
<location x="0" y="0" z="0"/>
<parameter name="jess" value="file:rxtest.clp"/>
<behaviours codebase="file:">
<sensor class="vwadm.awa.base.ACLSensor"/>
</behaviours>
</reteagent>

<reteagent name="txtest">
<location x="0" y="0" z="0"/>
<parameter name="jess" value="file:txtest.clp"/>
<behaviours codebase="file:">
<effector class="vwadm.awa.base.ACLEffector"/>
</behaviours>
</reteagent>

</MAS>
```

1.1.2 Annotated Jess rules for agent `txtest`

This is a demonstration sender agent. We begin with the standard import and defclass statements.

```
(import jess.*)
(import vwadm.awa.base.*)
(defclass vwadm_awa_base_ReteAgent ReteAgent)
(defclass vwadm_awa_base_ACLEffector ACLEffector)

(reset)
```

The first rule is a standard start-up rule.

```
(defrule start-up-rule
   (declare (salience 10))
```

```
   ?a <- (vwadm_awa_base_ReteAgent (OBJECT ?ag))
=>
   (set-multithreaded-io TRUE)
   ;(modify ?a (traceEnabled TRUE))
   (printout t "txtest initialised" crlf))
```

This demo sender agent sends one message only.

```
(defrule transmit-rule
   (initialised)
   (vwadm_awa_base_ReteAgent (OBJECT ?ag))
   ?eff <- (vwadm_awa_base_ACLEffector (OBJECT ?reference))
=>
   (bind ?a (new Action (?ag getName) ACTIVATE (create$ LISTENER)))
   (bind ?targ ((?ag getMAS) findAgent "rxtest"))
   (bind $?c (create$ ?a))
   (call ?reference setTarget ?targ)
   (modify ?eff (performative (get-member ACLEffector REQUEST))
                (content $?c))
   (call ?reference activate)
   (printout t "ACL INFORM tranmitted" crlf))
```

The performative is a `REQUEST` , and it gets the number of the `REQUEST` performative from the `ACLEFfector` class. The content it sends contains only one element: an instance of `Action` that it binds to variable `?a`. This `Action` is constructed with `actor` set to the name of the sender agent, `act` set to `"ACTIVATE"`, and `arguments` set to a list containing only the string `"LISTENER"`. The rule above sets up the variable `?a`, sets the target, performative and content, and then activates the effector. In more complex situations, the content may alternatively be set to an `Action` and a number of `Propositions` that provide further information, such as constraints, about the act.

1.1.3 Annotated Jess rules for agent `rxtest`

This is a demonstration receiver agent, whichbegins with the standard import and defclass statements.

```
(import jess.*)
(import vwadm.awa.base.*)

(defclass vwadm_awa_base_ReteAgent ReteAgent)
(defclass vwadm_awa_base_SenseData SenseData)
(defclass vwadm_awa_base_ACLSenseData ACLSenseData
   extends vwadm_awa_base_SenseData)
(defclass vwadm_awa_base_Content Content)
(defclass vwadm_awa_base_Action Action extends vwadm_awa_base_Content)
```

```
(defclass vwadm_awa_base_Proposition Proposition
   extends vwadm_awa_base_Content)
```

This agent defines one unordered fact to hold parsed ACL messages that have been received.

```
(deftemplate rxaction
   (slot actor (type LEXEME) (default nil))
   (slot act (type LEXEME) (default nil))
   (slot performative (type NUMBER) (default -1))
   (multislot arguments)
   (multislot propositions))

(defglobal ?*request* = (get-member vwadm.awa.base.ACLEffector REQUEST))

(reset)

(defrule start-up-rule
   (declare (salience 10))
   ?a <- (vwadm_awa_base_ReteAgent (OBJECT ?ag))
=>
   (set-multithreaded-io TRUE)
   (printout t "rxtest initialised" crlf))
```

The next pair of rules receive an ACL message and store it in the unordered fact just defined.

```
(defrule acl-receive-rule-1
   (initialised)
   (vwadm_awa_base_ReteAgent (OBJECT ?ag))
   ?f <- (vwadm_awa_base_ACLSenseData
               (performative ?perf)
               (content ?c $?p)
               (sender ?sender))
   (test (instanceof ?c "Action"))
=>
   (assert (rxaction (actor (?c getActor))
                     (act (?c getAct))
                     (performative ?perf)
                     (arguments (?c getArguments))
                     (propositions $?p)))
   (retract ?f)
)

(defrule acl-receive-rule-2
   (declare (salience -10))
```

```
20      ?f <- (vwadm_awa_base_ACLSenseData)
21    =>
22      (retract ?f)
23    )
```

This pair of rules interpret a parsed message. For this demo, it looks for a specific encoded message.

```
1   (defrule acl-parse-citizen-rule-1
2      ?f <- (rxaction (actor "txtest")
3                      (act "ACTIVATE")
4                      (performative ?perf&:(= ?*request* ?perf))
5                      (arguments "LISTENER" $?))
6   =>
7      (printout t "REQUEST from txtest: ACTIVATE LISTENER" crlf)
8      (retract ?f)
9   )
10
11  (defrule acl-parse-rule-2
12     (declare (salience -10))
13     ?f <- (rxaction)
14  =>
15     (retract ?f)
16  )
```

1.1.4 Running the demsonstration

Running the demonstration sender and receiver agents results in the following being printed to the standard output:

```
rxtest initialised
txtest initialised
Agent "rxtest" entered into MAS "txrxtest"
INFO: Agent rxtest avatar is OK at location 0N 0W height 0
INFO: Agent txtest is OK
ACL INFORM tranmitted
REQUEST from txtest: ACTIVATE LISTENER
```

which shows that the message is sent and received correctly.

1.2 Demonstration of a request protocol

1.2.1 Protocols

A protocol is a pattern for interaction between agents. It defines the sequence of messages that agents should send to each other. A protocol, a communication language

(such as the FIPA ACL), and an ontology together define what messages agents can send each other.

In this section, a demonstration is given of a request protocol. A request protocol allows one agent to request another to perform some action and the receiving agent to perform the action or reply, in some way, that it cannot FIPA (2002). Assume for the sake of example that a concierge agent called `Jack` desires that slides be displayed for use during a meeting. `Jack` knows that a wall agent called `Wall1` can display HTML as graffiti if it is given the URL of a set of HTML slides. `Jack` therefore instantiates a request protocol to achieve this. This protocol is shown in Figure 1 as initiating a `REQUEST` from Jack to Wall1 to display the slides referred to by the URL on a PROJECTOR. The tokens are all strings in these Petri nets only for simplicity of the simulation; in the next section, a Jess implementation is discussed.

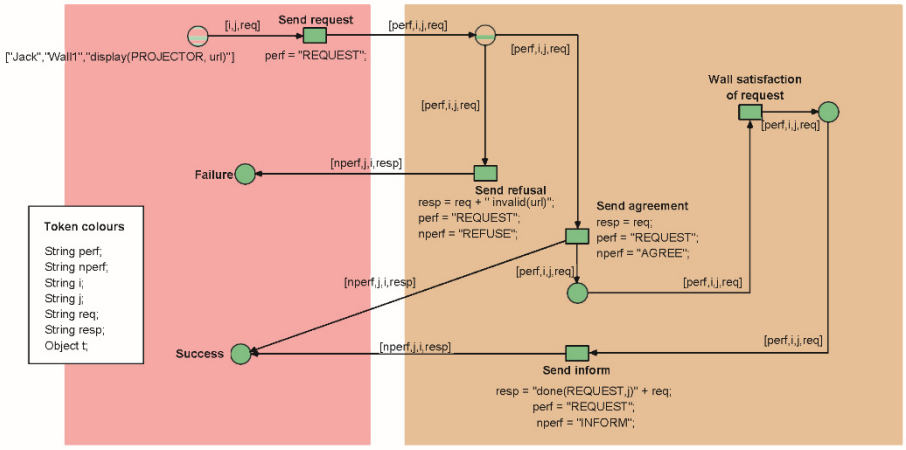


Figure 1: Example request interaction protocol Petri net initial state: Agent `Jack` sends a `REQUEST` performative to agent `Wall1` with content display(PROJECTOR,url). `Jack` is indicated by the left hand shaded background; `Wall1` is indicated by the right hand shaded background. Petri net places containing tokens are shown highlighted.

Returning to the example, `Wall1` receives the `REQUEST` message and interprets it as is shown in Figure 2. If `Wall1` agrees to the request, it returns an `AGREE` message (Figure 3), takes actions so as to satisfy that request (Figure 4), and returns an `INFORM` message to indicate completion (Figure 5). `Wall1` satisfies the request by redesigning the wall entity that it represents so as to display graffiti containing formatted slides (Figure 6.).

If for some reason the request cannot be satisfied, then a *refuse* message would be returned. Figure 7 shows an example where the specified URL is invalid.

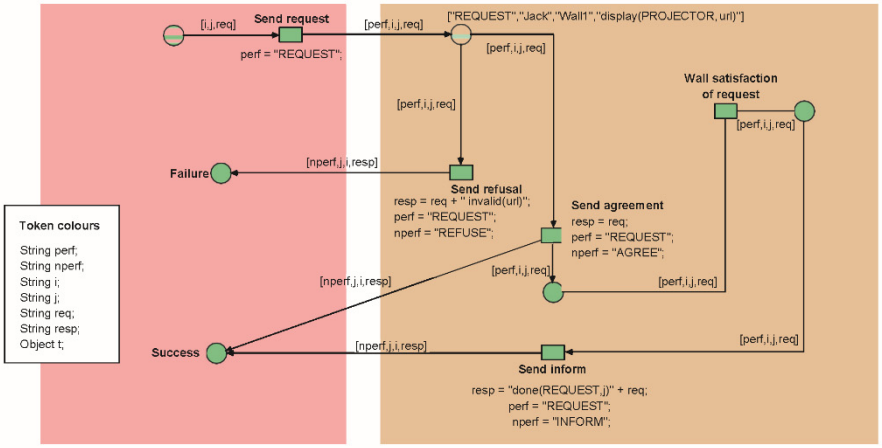


Figure 2: Figure 1 after firing the "Send request" transition: agent `Wall1` receives the `REQUEST`.

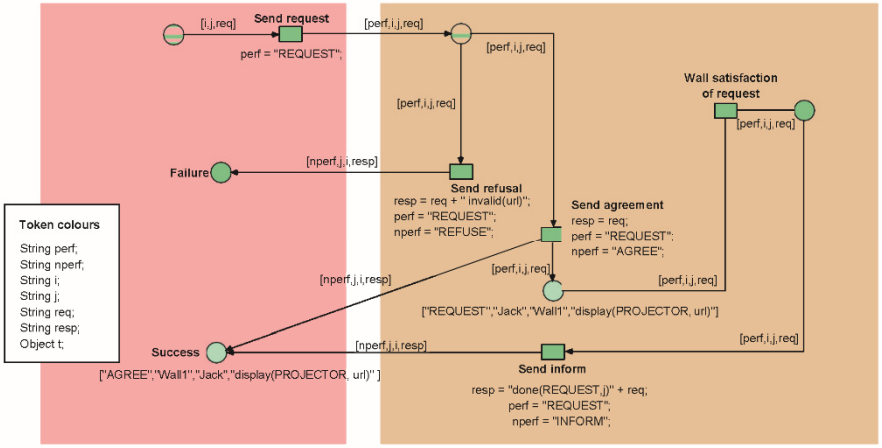


Figure 3: Figure 2 after firing the "Send agreement" transition: `Wall1` sends an `AGREE` message to `Jack`.

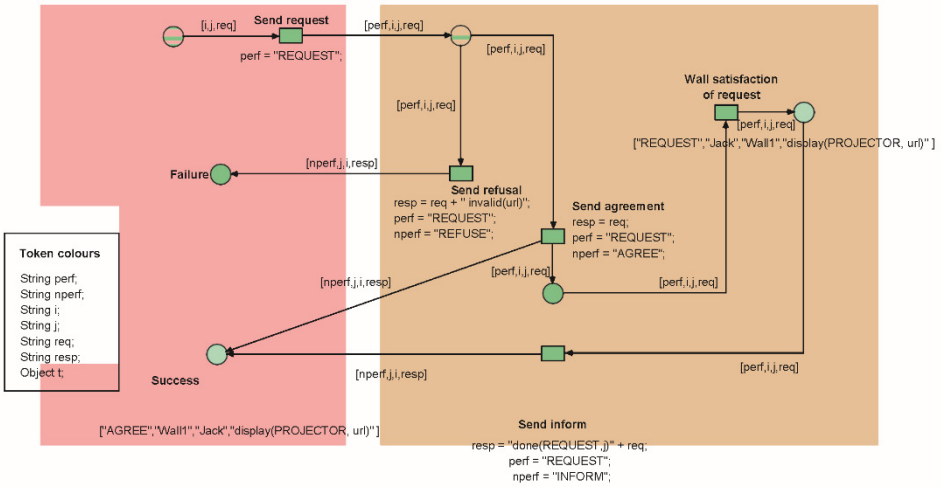


Figure 4: Figure 3 after firing the "Wall satisfaction of request" transition: agent `Wall1` activates actions so as to satisfy the request from `Jack`.

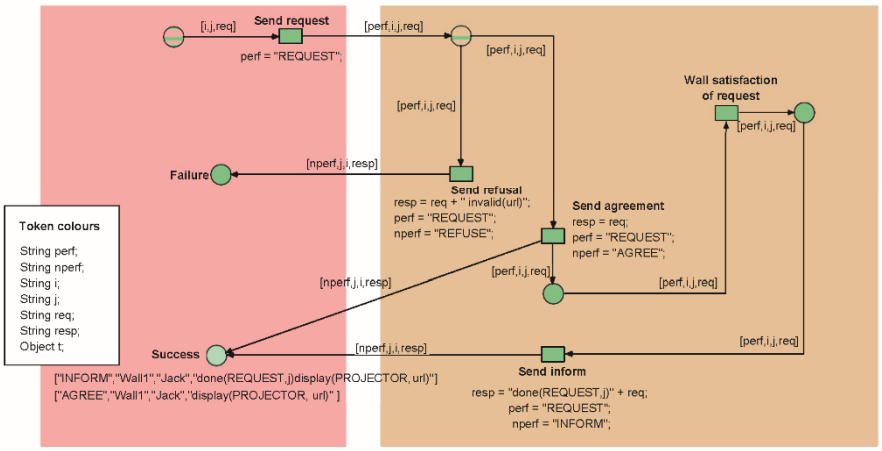


Figure 5: Figure 4 after firing the "Send inform" transition": `Wall1` sends an `INFORM` to `Jack` to inform it that the request has been satisfied.

Figure 6: Writing graffiti on a wall to display information.

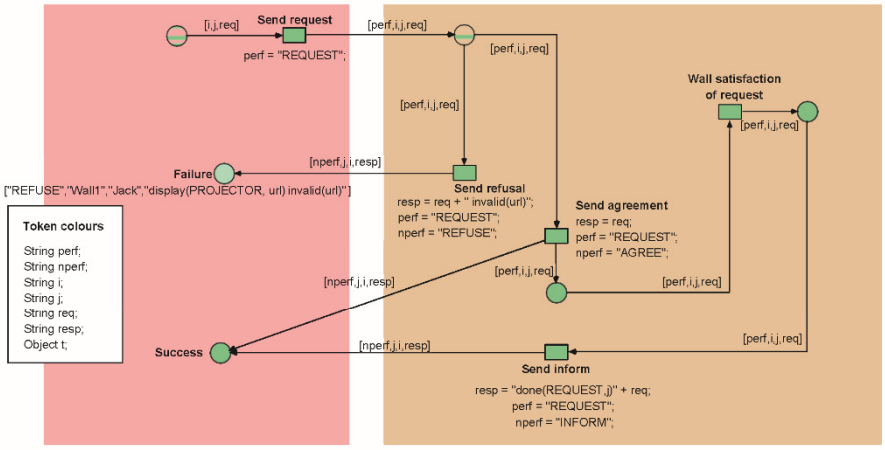


Figure 7: Alternative scenario to Figures 2 through 5: agent Wall1 sends a `REFUSE` message to jack to inform it that the request will not be satisfied because the url is invalid.

1.2.2 Configuration

request.xml

```
<?xml version="1.0" ?>
<!DOCTYPE MAS SYSTEM "http://mojo.csd.anglia.ac.uk/william/MAS.dtd">
<MAS>

<connection class="vwadm.awa.base.Connection">
   <parameter name="avatarname" value="txrxtest"/>
</connection>

<reteagent name="Wall1">
<parameter name="jess" value="file:rxrequest.clp"/>
<location x="0" y="0" z="0"/>
<behaviours codebase="file:">
<sensor class="vwadm.awa.base.ACLSensor"/>
<effector class="vwadm.awa.base.ACLEffector"/>
</behaviours>
</reteagent>

<reteagent name="Jack">
<parameter name="jess" value="file:txrequest.clp"/>
<location x="0" y="0" z="0"/>
<behaviours codebase="file:">
<sensor class="vwadm.awa.base.ACLSensor"/>
<effector class="vwadm.awa.base.ACLEffector"/>
</behaviours>
</reteagent>

</MAS>
```

1.2.3 Annotated Jess rules for agent `Jack`

This is the initiator of the interaction, `Jack`.

```
(import jess.*)
(import vwadm.awa.base.*)
(defclass vwadm_awa_base_ReteAgent ReteAgent)
(defclass vwadm_awa_base_SenseData SenseData)
(defclass vwadm_awa_base_ACLSenseData ACLSenseData
   extends vwadm_awa_base_SenseData)
(defclass vwadm_awa_base_ACLEffector ACLEffector)

(deffacts initialfact
   (sendmessage))
(reset)
```

The first rule is the standard start-up rule.

```
(defrule start-up-rule
   (declare (salience 10))
   ?a <- (vwadm_awa_base_ReteAgent (OBJECT ?ag))
=>
   (set-multithreaded-io TRUE)
   ;(modify ?a (traceEnabled TRUE))
   (printout t (?a getName) " initialised" crlf))

(deffunction MAIN::terminate ())
```

Jack sends only one message in this demonstration.

```
(defrule transmit-rule
   (initialised)
   ?f <- (sendmessage)
   (vwadm_awa_base_ReteAgent (OBJECT ?ag))
   ?eff <- (vwadm_awa_base_ACLEffector (OBJECT ?reference))
=>
   (bind ?a (new Action "Wall1"
      DISPLAY (create$
        PROJECTOR
        "http://mojo.csd.anglia.ac.uk/william/AWbot/demos/html/slides.txt")))
   (bind ?targ ((?ag getMAS) findAgent "Wall1"))
   (bind $?c (create$ ?a))
   (call ?reference setTarget ?targ)
   (modify ?eff (performative (get-member ACLEffector REQUEST))
                (content $?c))
   (call ?reference activate)
   (printout t "ACL REQUEST transmitted from " (?ag getName) " to Wall1" crlf)
   (retract ?f)
)
```

Jack needs to receive replies from Wall1, but in this demo, nothing is done with them.

```
(defrule acl-receive-rule
   (declare (salience -10))
   ?f <- (vwadm_awa_base_ACLSenseData)
=>
   (retract ?f)
)
```

1.2.4 Annotated Jess rules for agent `Wall1`

This demo of Wall1 always replies to a request with success, but it doesn't actually do anything - it just pretends that it satisfies the request.

The protocol in this example is implemented entirely in Jess. In order that many different protocols can be implemented by the one agent we encode the Petri nets as Jess facts that collectively describe a partial order plan. A generic set of plan activation rules then activate plan steps whenever the preconditions for the step are satisfied. A particular `REQUEST` is then handled by one interpretation rule instantiating the plan.

```
1  (import jess.*)
2  (import vwadm.awa.base.*)
3
4  (defclass vwadm_awa_base_ReteAgent ReteAgent)
5  (defclass vwadm_awa_base_SenseData SenseData)
6  (defclass vwadm_awa_base_ACLSenseData ACLSenseData
7     extends vwadm_awa_base_SenseData)
8  (defclass vwadm_awa_base_Content Content)
9  (defclass vwadm_awa_base_Action Action extends vwadm_awa_base_Content)
10 (defclass vwadm_awa_base_Proposition Proposition extends
11 vwadm_awa_base_Content)(defclass vwadm_awa_base_ACLEffector ACLEffector)
12
13 ;This is what to do at step 'no'
14 (deftemplate protocol-step
15    (slot plan)
16    (slot no (type NUMBER) (default 0))
17    (slot op (type LEXEME) (default nil))
18    (multislot params (default (create$))))
19
20 ;One causal link in a protocol - for a conjunction there will be one
21 ;link fact per conjunctive term
22 (deftemplate protocol-link
23    (slot plan)
24    (slot from (type NUMBER) (default 0))
25    (slot to   (type NUMBER) (default 0))
26    (multislot preconds))
27
28 ;One ordering constraint in a protocol
29 (deftemplate protocol-order
30    (slot plan)
31    (slot from (type NUMBER) (default 0))
32    (slot to   (type NUMBER) (default 0)))
33
34 (deftemplate comm
35    (slot plan)
36    (slot op (type LEXEME))
37    (multislot params (default (create$))))
38
39 (defglobal ?*request* = (get-member vwadm.awa.base.ACLEffector REQUEST))
```

```
40
41  (reset)
```

The deftemplate `protocol-step` is one step in the plan. Slot `op` is an action for which there is one or more action rules, and slot `param` contains parameters for that action. Deftemplate `protocol-order` is an ordering constraint: it says that step `from` must activate before step `to`. Deftemplate `protocol-link` says that the activation of step `from` provides preconditions `preconds` required by step `to`.

```
1   (defrule start-up-rule
2      (declare (salience 10))
3      ?a <- (vwadm_awa_base_ReteAgent (OBJECT ?ag))
4   =>
5      (set-multithreaded-io TRUE)
6      ;(modify ?a (traceEnabled TRUE))
7      (printout t (?a getName) " initialised" crlf))
8
9   (deffunction MAIN::terminate ())
```

The next rules interpret received `REQUEST` ACL messages. In this demo, we only look for `REQUEST`s. When the agent receives one it asserts a set of `comm` facts that describe the new communication: `ACTOR` is the agent that the `REQUEST` is target at, `SENDER` is the agent that initiated this interaction, `ACT` is the particular act that is being requested, `PERF` is the performative (that is, a `REQUEST`), `ARGS` are optional parameters of the `ACT`, and `STATE` is the current state of the protocol.

```
1   (defrule acl-receive-rule-1
2      (initialised)
3      (vwadm_awa_base_ReteAgent (OBJECT ?ag))
4      ?f <- (vwadm_awa_base_ACLSenseData
5                  (performative ?perf)
6                  (content ?c $?)
7                  (sender ?sender))
8      (test (instanceof ?c "Action"))
9      (test (= ?*request* ?perf))
10  =>
11     (bind ?pl (gensym*))
12     (assert (comm (plan ?pl) (op ACTOR) (params (?c getActor)))
13             (comm (plan ?pl) (op SENDER) (params ?sender))
14             (comm (plan ?pl) (op ACT) (params (?c getAct)))
15             (comm (plan ?pl) (op PERF) (params REQUEST))
16             (comm (plan ?pl) (op ARGS) (params (?c getArguments)))
17             (comm (plan ?pl) (op STATE) (params START))
18     )
19
```

```
   (retract ?f)
)

(defrule acl-receive-rule-2
   (declare (salience -10))
   ?f <- (vwadm_awa_base_ACLSenseData)
=>
   (retract ?f)
)
```

Next is a rule that instantiates the plan that will control this agent's end of the protocol. For this example it only looks for the `DISPLAY PROJECTOR ?url` message.

```
(defrule instantiate-request-protocol
   (comm (plan ?pl) (op PERF) (params REQUEST))
   (comm (plan ?pl) (op ACT)  (params "DISPLAY"))
   (comm (plan ?pl) (op ARGS) (params "PROJECTOR" ?url))
   ?f <- (comm (plan ?pl) (op STATE) (params START))
   (not (protocol-step (plan ?pl)))
=>
   (modify ?f (params STARTED))

   (assert (protocol-step (plan ?pl) (no 1)
                          (op START) (params ?pl)))

   (assert (protocol-link (plan ?pl) (from 1) (to 3)
                          (preconds STATE REJECTED)))
   (assert (protocol-step (plan ?pl) (no 3)
                          (op SEND) (params REFUSAL ?pl)))

   (assert (protocol-link (plan ?pl) (from 1) (to 4)
                          (preconds STATE ACCEPTED)))
   (assert (protocol-step (plan ?pl) (no 4)
                          (op SEND) (params AGREEMENT ?pl)))

   (assert (protocol-link (plan ?pl) (from 1) (to 5)
                          (preconds STATE ACCEPTED)))
   (assert (protocol-order (plan ?pl) (from 4) (to 5)))
   (assert (protocol-step (plan ?pl) (no 5)
                          (op SATISFY) (params REQUEST ?pl)))

   (assert (protocol-link (plan ?pl) (from 3) (to 6)
                          (preconds STATE REFUSED)))
   (assert (protocol-step (plan ?pl) (no 6)
                          (op CHANGE-STATE) (params FINISHED ?pl)))

   (assert (protocol-link (plan ?pl) (from 4) (to 7)
                          (preconds STATE ACHIEVED)))
   (assert (protocol-step (plan ?pl) (no 7)
```

```
                              (op CHANGE-STATE) (params FINISHED ?pl)))

   (assert (protocol-link (plan ?pl) (from 6) (to 8)
                          (preconds STATE FINISHED)))
   (assert (protocol-link (plan ?pl) (from 7) (to 8)
                          (preconds STATE FINISHED)))
   (assert (protocol-step (plan ?pl) (no 8)
                          (op FINISH) (params ?pl)))
)


(defrule instantiate-protocol-default1
   (declare (salience -5))
   (comm (plan ?pl) (op PROTOCOL) (params START))
   ?comm <- (comm (plan ?lbl) (op ~PROTOCOL))
=>
   (retract ?comm)
)
(defrule instantiate-protocol-default2
   (declare (salience -10))
   ?comm <- (comm (plan ?lbl) (op PROTOCOL) (params START))
=>
   (retract ?comm)
)
```

This next set of rules are generic plan activation rules. They could be applied to other plans of the agent other than for communication protocols.

Rule `select-action-candidates-rule` asserts one temporary fact `candidate` for each potentially activatable step.

Rule `select-action-pruning-rule-1` retracts a candidate fact if there is an unsatisfied causal link to this step.

Rule `select-action-pruning-rule-2` retracts a candidate fact if there is an ordering constraint from a step that isn't satisfied.

Rule `select-candidate-rule` selects, from one of the remaining candidate, one to activate; it asserts an action activation fact `action` for this.

The cleanup rules cleanup after the candidate selection finishes, and clean up the protocol plan facts after the protocol finishes.

```
(defrule select-action-candidates-rule
   (declare (salience 5))
   (comm (plan ?lbl))
   (not (action ?pl $?))
   (protocol-step (plan ?pl) (no ?no) (op ?op) (params ?opr $?param))
=>
   (assert (candidate ?pl ?no))
)

```

```
(defrule select-action-pruning-rule-1
   ?c <- (candidate ?pl ?no)
   (protocol-link (plan ?pl) (to ?no) (preconds ?opr $?pre))
   (not (comm (plan ?pl) (op ?opr) (params $?pre)))
=>
   (retract ?c)
)

(defrule select-action-pruning-rule-2
   ?c <- (candidate ?pl ?no)
   (protocol-order (plan ?pl) (from ?fr) (to ?no))
   (protocol-step (plan ?pl) (no ?fr))
=>
   (retract ?c)
)

(defrule select-candidate-rule
   (declare (salience -5))
   (not (action ?pl $?))
   (candidate ?pl ?no)
   (protocol-step (plan ?pl) (no ?no) (op ?op) (params $?param))
=>
   (assert (action ?pl ?no ?op $?param))
)

(defrule cleanup-candidates-rule-1
   (declare (salience 10))
   (action ?pl ?no $?)
   ?c <- (candidate ?pl ?no)
=>
   (retract ?c)
)
(defrule cleanup-candidates-rule-2
   (declare (salience 10))
   (action ?pl ?no $?)
   ?f <- (protocol-step (plan ?pl) (no ?no))
=>
   (retract ?f)
)
(defrule cleanup-candidates-rule-3
   (declare (salience 10))
   (action ?pl ?no $?)
   ?f <- (protocol-link (plan ?pl) (to ?no))
=>
   (retract ?f)
)
(defrule cleanup-candidates-rule-4
   (declare (salience 10))
   (action ?pl ?no $?)
   ?f <- (protocol-order (plan ?pl) (to ?no))
=>
```

```
   (retract ?f)
)

(defrule remove-protocol-1
   (declare (salience -10))
   (comm (plan ?pl) (op PROTOCOL) (params FINISH))
   ?f <- (comm (plan ?lbl) (op ~PROTOCOL))
=>
   (retract ?f)
)

(defrule remove-protocol-2
   (declare (salience -10))
   (comm (plan ?pl) (op PROTOCOL) (params FINISH))
   ?f <- (protocol-step (plan ?pl))
=>
   (retract ?f)
)
(defrule remove-protocol-3
   (declare (salience -10))
   (comm (plan ?pl) (op PROTOCOL) (params FINISH))
   ?f <- (protocol-link (plan ?pl))
=>
   (retract ?f)
)
(defrule remove-protocol-4
   (declare (salience -10))
   (comm (plan ?pl) (op PROTOCOL) (params FINISH))
   ?f <- (protocol-order (plan ?pl))
=>
   (retract ?f)
)
(defrule remove-protocol-5
   (declare (salience -20))
   ?comm <- (comm (plan ?lbl) (op PROTOCOL) (params FINISH))
=>
   (retract ?comm))
```

The rest of the rules are one per action.

```
(defrule FINISH-operator
   ?a <- (action ? ?no FINISH ?pl)
   ?c <- (comm (plan ?pl) (op STATE))
=>
   (retract ?a)
   (printout t "FINISH operator" crlf)
   (modify ?c (params FINISH))
)

```

```
10
11  (defrule START-operator-1
12     ?f <- (action ?pl ?no START $?)
13     (comm (plan ?pl) (op ACT) (params "DISPLAY"))
14     (comm (plan ?pl) (op PERF) (params REQUEST))
15     (comm (plan ?pl) (op ARGS) (params "PROJECTOR" ?url))
16     ?c <- (comm (plan ?pl) (op STATE))
17  =>
18     (retract ?f)
19     (printout t "START operator for REQUEST DISPLAY PROJECTOR ?url" crlf)
20     (modify ?c (params ACCEPTED))
21  )
22  (defrule START-operator-2
23     (declare (salience -1))
24     ?f <- (action ?pl ?no START)
25     ?c <- (comm (plan ?pl) (op STATE))
26  =>
27     (retract ?f)
28     (printout t "START operator default" crlf)
29     (modify ?c (params REJECTED))
30  )
31
32  (defrule REFUSAL-operator
33     ?f <- (action ? ?no SEND REFUSAL ?pl)
34     (comm (plan ?pl) (op ACTOR) (params ?a))
35     (comm (plan ?pl) (op SENDER) (params ?s))
36     ?c <- (comm (plan ?pl) (op STATE))
37  =>
38     (retract ?f)
39     (modify ?c (params REFUSED))
40     (printout t "REFUSAL sent from " ?a " to " ?s crlf)
41  )
42
43  (defrule AGREEMENT-operator
44     (vwadm_awa_base_ReteAgent (OBJECT ?ag))
45     ?eff <- (vwadm_awa_base_ACLEffector (OBJECT ?reference))
46     ?f <- (action ? ?no SEND AGREEMENT ?pl)
47     (comm (plan ?pl) (op ACTOR) (params ?a))
48     (comm (plan ?pl) (op SENDER) (params ?s))
49     (comm (plan ?pl) (op ACT) (params ?act))
50     (comm (plan ?pl) (op ARGS) (params $?args))
51     ?c <- (comm (plan ?pl) (op STATE))
52  =>
53     (retract ?f)
54     (bind ?targ ((?ag getMAS) findAgent ?s))
55     (bind ?a (new Action (?targ getName)
56        AGREE (create$ ?act ?args)))
57     (bind $?c (create$ ?a))
58     (call ?reference setTarget ?targ)
59     (modify ?eff (performative (get-member ACLEffector INFORM))
60                  (content $?c))
```

```
61      (call ?reference activate)
62      (printout t "AGREEMENT sent from " ?a " to " ?s crlf)
63   )
64
65   ;In this demo we don't actually satisfy the request:
66   ;we just pretend that we do
67   (defrule SATISFY-operator-1
68      ?f <- (action ? ?no SATISFY REQUEST ?pl)
69      (comm (plan ?pl) (op ACTOR) (params ?a))
70      (comm (plan ?pl) (op SENDER) (params ?s))
71      (comm (plan ?pl) (op ACT) (params ?act))
72      (comm (plan ?pl) (op ARGS) (params $?args))
73      ?c <- (comm (plan ?pl) (op STATE))
74   =>
75      (retract ?f)
76      (printout t "SATISFY REQUEST " ?act $?args " sent from " ?a
77                                                      " to " ?s crlf)
78      (modify ?c (params ACHIEVED))
79   )
80
81   ;This is a NOOP
82   (defrule CHANGE-STATE-operator-1
83      ?f <- (action ? ?no CHANGE-STATE ?newstate ?pl)
84      ?c <- (comm (plan ?pl) (op STATE))
85   =>
86      (retract ?f)
87      (printout t "CHANGE-STATE operator" crlf)
88      (modify ?c (params ?newstate))
89   )
90
```

2 Dynamically creating an agent

In this section, a demonstration is given on how to have one agent dynamically create another. The creating agent is called "creator", the created agent is called "disciple".

2.1 Configuration

As "creator" will create "disciple", only "creator" needs to be in the XML.

creator.xml

```
1   <?xml version="1.0" ?>
2   <!DOCTYPE MAS SYSTEM "http://mojo.csd.anglia.ac.uk/william/MAS.dtd">
3   <MAS>
4
5   <connection class="vwadm.awa.base.AWConnection">
6      <parameter name="domain" value="auth.activeworlds.com"/>
7      <parameter name="port" value="6670"/>
```

```
    <parameter name="avatarType" value="4"/>
    <parameter name="avatarname" value="creation"/>
    <parameter name="worldname" value="agent"/>
</connection>

<reteagent name="creator">
<parameter name="jess" value="file:creator.clp"/>
<location x="-1000" y="0" z="0"/>
<behaviours codebase="file:">
</behaviours>
</reteagent>

</MAS>
```

In this demonstration, the creator does nothing except create the disciple, so it has no sensors or effectors.

2.2 Annotated Jess rules for agent `creator`

The following begins with the usual import and defclass statements.

```
(import jess.*)
(import vwadm.awa.base.*)

(clear)

(defclass vwadm_awa_base_Agent Agent)
(defclass vwadm_awa_base_ReteAgent ReteAgent extends vwadm_awa_base_Agent)
(defclass vwadm_awa_base_DynamicReteAgent DynamicReteAgent
   extends vwadm_awa_base_Agent)

(reset)
```

This agent contains only one rule:

```
(defrule start-up-rule
   (declare (salience 10))
   ?a <- (vwadm_awa_base_ReteAgent (OBJECT ?ag))
   (initialised)
=>
   (set-multithreaded-io FALSE)

   ;Create a bag of parameters for DynamicReteAgent
   (bind ?bg (bag create agentBag))

```

```
    ;Set parameters required for DynamicReteAgent
    (bag set ?bg name "disciple")
    (bag set ?bg jess "file:disciple.clp")
    (bag set ?bg x "200")
    (bag set ?bg y "0")
    (bag set ?bg z "200")

    ;Set optional parameters
    (bag set ?bg yaw "1800")

    ;Set a demo parameter for disciple
    (bag set ?bg anotherParameter "a parameter string")

    ;Create a DynamicReteAgent
    (bind ?soc (call ?ag getMAS))
    (bind ?dynagent (new DynamicReteAgent ?soc ?bg (engine)))

    ;Configure one sensor
    (bind ?sbg (bag create sensorBag1))
    (bag set ?sbg width 500)
    (bag set ?sbg height 500)
    (call ?dynagent addSensor
                    "vwadm.awa.base.AW3DObjectSensor"
                    ?sbg
                    (engine))


    ;Configure one effector - chat effector requires no parameters
    (call ?dynagent addEffector
                    "vwadm.awa.base.AWChatEffector"
                    nil
                    (engine))

    ;and start the agent
    (call ?dynagent addToMAS)
)
```

The antecedent side of the above rule does the following:

1. Create a bag of parameters for the agent. As with any agent, you can pass it an arbitrary number of parameters. From the java side the agent sees these as an instance of class `java.util.Hashtable`. From the Jess side these are a `bag` - Jess and the java class `vwadm.awa.base.DynamicReteAgent` together handle any conversions required. The Jess `bag` command lets you manipulate hash tables (also called maps or associative arrays).

```
    (bind ?bg (bag create agentBag))
```

2. Set in the bag those parameters that are required for the agent. The following are the minimum:

```
(bag set ?bg name "disciple")
(bag set ?bg jess "file:disciple.clp")
(bag set ?bg x "200")
(bag set ?bg y "0")
(bag set ?bg z "200")
```

3. Set any optional parameters (such as yaw) and any that are required to configure your particular agent. One parameter called "anotherParameter" with value "a parameter string" has been added for demonstration purposes only.

```
(bag set ?bg yaw "1800")
(bag set ?bg anotherParameter "a parameter string")
```

An example use for extra parameters is as follows. Suppose that you have one tradesman agent that builds walls. It builds walls by dynamically creating wall agents. Rather than have a different CLP file for each wall agent, you would have one file "wall.clp" and parameters with which to instantiate it (such as RWX model name and texture).

4. Once you have a bag of agent properties you can create the agent. A new java class `vwadm.awa.base.DynamicReteAgent` has been written; it extends the usual agent class `vwadm.awa.base.ReteAgent`. `vwadm.awa.base.DynamicReteAgent` includes methods to simplify the dynamic addition of agents to a MAS at run time (that is, after all agents have been configured and initialised from the XML) from Jess. In other respects `vwadm.awa.base.DynamicReteAgent` behaves just like `vwadm.awa.base.ReteAgent`.

```
(bind ?soc (call ?ag getMAS))
(bind ?dynagent (new DynamicReteAgent ?soc ?bg (engine)))
```

5. You call one method to add each sensor and one method to add each effector. Both take as a parameter a bag of parameters to configure the sensor or effector.

```
(bind ?sbg (bag create sensorBag1))
(bag set ?sbg width 500)
(bag set ?sbg height 500)
(call ?dynagent addSensor
```

```
                    "vwadm.awa.base.AW3DObjectSensor"
                    ?sbg
                    (engine))

    ;chat effector requires no parameters, hence nil
    (call ?dynagent addEffector
                    "vwadm.awa.base.AWChatEffector"
                    nil
                    (engine))
```

6. And finally, we start the agent

```
    (call ?dynagent addToMAS)
```

2.3 Annotated Jess rules for agent `disciple`

The following begins with the usual import and defclass statements.

```
(import jess.*)
(import vwadm.awa.base.*)

(clear)

(defclass vwadm_awa_base_Agent Agent)
(defclass vwadm_awa_base_ReteAgent ReteAgent extends vwadm_awa_base_Agent)
(defclass vwadm_awa_base_DynamicReteAgent DynamicReteAgent
   extends vwadm_awa_base_Agent)
(defclass vwadm_awa_base_AWChatEffector AWChatEffector)
(defclass vwadm_awa_base_SenseData SenseData)
(defclass vwadm_awa_base_Object3DSenseData Object3DSenseData
   extends vwadm_awa_base_SenseData)
(defclass vwadm_awa_base_AddedObject3DSenseData AddedObject3DSenseData
   extends vwadm_awa_base_Object3DSenseData)


(reset)
```

For this example we retrieve the parameter "anotherParameter" from the bag of parameters and store in in a defglobal.

```
(defglobal ?*anotherParameter* = nil)

(defrule start-up-rule
```

```
    (declare (salience 10))
    (initialised)
    ?a <- (vwadm_awa_base_ReteAgent (OBJECT ?ag))
=>
    (set-multithreaded-io TRUE)

    (printout t (ppdeftemplate vwadm_awa_base_ReteAgent) crlf)
    (modify ?a (traceEnabled TRUE))

    (bind ?*anotherParameter*
          (?ag getConfigurationParameter "anotherParameter"))
)
```

For the sake of this demo we have the agent use its chat effector to print out the value of `?*anotherParameter*`.

```
(defrule chat-action-rule
    (initialised)
    ?eff <- (vwadm_awa_base_AWChatEffector (OBJECT ?reference))
=>
    (modify ?eff (msg (str-cat "DynamicReteAgent configured with parameter: "
                               ?*anotherParameter*) )
                 (msgType (get-member MAS CHAT_SAID)))
    (call ?reference activate)
)
```

And to show that the sensor is configured, we have two rules.

```
(defrule sense-rule-1
    (initialised)
    ?f <- (vwadm_awa_base_AddedObject3DSenseData (model ?m)
                                                 (objectLocn ?locn))
=>
    (printout t "Object " ?m " found at " (?locn toString) crlf)
    (retract ?f)
)

(defrule sense-rule-2
    (declare (salience -10))
    (initialised)
    ?f <- (vwadm_awa_base_SenseData)
=>
    (retract ?f)
)
```

3 Writing Sensors and Effectors

Here, an explanation is given for writing your own sensors.

3.1 Extending the Sensor Class

Sensors are written in Java and implement an observer pattern. Some sensors also have methods written in C that are called via a Java Native Interface (JNI). Figure 8 shows an object diagram of the main classes involved.

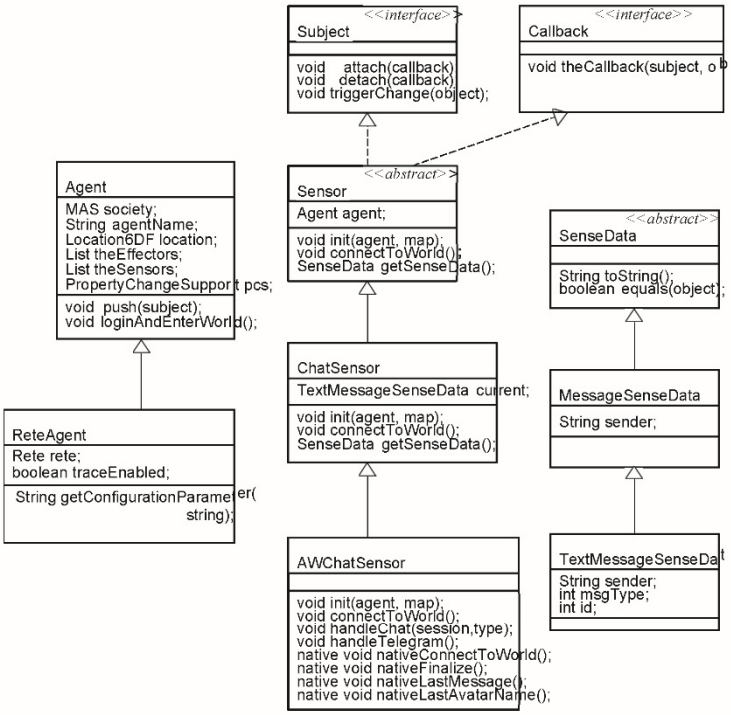


Figure 8: Main AWAgent classes involved in writing a sensor. Only those classes, interfaces, methods and properties necessary to understand this section are shown.

In this discussion, an assumption is being made that sensors are being written for use by `ReteAgent`s. Shown in Figure 8 is the abstract sensor class `Sensor` which all sensors must extend. Sensor classes should override the three methods shown: `init`, `connectToWorld` and

`getSenseData`. The observer pattern works by having an agent, be it an `Agent` or a `ReteAgent`, register interest in observing the sensor and having the sensor notify the any registered observers when new sense-data arrives. This allows sensors to operate asynchronously to the agents and in an event-driven manner. It works as follows:

. For each sensor in the XML configuration of an agent the class loader loads the class specified by the `class` attribute. This means that any sensor class in any jar file of the classpath can be instantiated. The agent then calls the `void init(Agent a, Map params)` method with the constructing agent set to `a` and with `params` set to a `java.util.HashMap` containing the contents of each XML `parameters` element of that sensor. For example, `AW3DObjectEffector` expects the following in the XML:

```
<sensor class="vwadm.awa.base.AW3DObjectSensor">
   <parameter name="width" value="500"/>
   <parameter name="height" value="500"/>
</sensor>
```

so that `params` will be set to the following pairs:

width 1→ 500
height 1→ 500

If you need to read any configuration for your sensor, then you should override the `init` method as follows:

```
public void init(Agent _agent, Map params)
{
   super.init(_agent, params);
   //extract config from params and save as instance properties
}
```

. The sensor is added to the agent property `theSensors` and the agent is registered as an observer of the sensor.

. Method `connectToWorld` is called on each sensor. You should override this if your sensor needs to do anything to attach to a world. For example, `connectToWorld` in `AWChatSensor` uses a JNI native method `nativeConnectToWorld` to have the AW SDK event handler call `handleChat` when new chat data arrives. See Section **3.2** for further details.

- The AW SDK is event driven. When new chat data arrives a JNI native method calls `handleChat`. Method `handleChat` constructs a new `TextMessaeSenseData` and for each registered observer it calls `theCallback`. This results in method `push` being called on each agent.
- In `ReteAgent` (`Agent` is similar) the `push` method calls `getSenseData` on the pushing sensor. Therefore, you should override `getSenseData` to return whatever sense data is appropriate.

In summary, then, for simple sensors you need to extend override class `Sensor`. You should override `init` if your sensor requires XML configuration parameters. You should override `connectToWorld` if your sensor needs to connect with external resources such as AW or an external databas. You should override `getSenseData` to return the appropriate sense data.

Every sensor has to construct one or more kinds of sense data. Each of these is a Java bean that must have a `toString` method, an `equals` method (you *must* implement this properly for Jess to work), and one Java bean property (with corresponding get and set methods) for each slot that you require in the corresponding Jess fact.

3.2 Hints on Using JNI

The AW SDK is a C language library that is accessed by sensors and effectors via a JNI. See (Liang 1999) for details of the JNI. Writing sensors and effectors that access such native libraries consists of the following steps.

1. Declare one or more native methods. For example, in `AWChatSensor` there are the following declarations:

```
native void nativeConnectToWorld();
native void nativeFinalize();
native String nativeLastMessage();
native String nativeLastAvatarName();
```

2. Use `javac` to compile your sensor Java class, and then use `javah -jni` to generate a C language header file. For the `nativeConnectToWorld` method this results in the following C function signature:

```
/*
 * Class:     vwadm_awa_base_AWChatSensor
 * Method:    nativeConnectToWorld
 * Signature: ()V
 */
JNIEXPORT void JNICALL
```

```
Java_vwadm_awa_base_AWChatSensor_nativeConnectToWorld
  (JNIEnv *, jobject);
```

3. Write a C language implementation of your native methods. For `nativeConnectToWorld` this is as follows:

```
JNIEXPORT void JNICALL
Java_vwadm_awa_base_AWChatSensor_nativeConnectToWorld(
   JNIEnv *env, jobject obj)
{
   jobject temp = (*env)->FindClass(env, "vwadm/awa/base/AWChatSensor");
   if (temp != NULL && AWChatSensorClass == NULL)
      AWChatSensorClass = (*env)->NewWeakGlobalRef(env, temp);

   if (AWChatSensorSelf == NULL ||
       !(*env)->IsSameObject(env, AWChatSensorSelf, obj))
      AWChatSensorSelf = (*env)->NewGlobalRef(env, obj);

   MID_AWChatSensor_handleChat =
      (*env)->GetMethodID(env,
                          AWChatSensorClass,
                          "handleChat",
                          "(II)V");

   int s = OBTAINLOCK(env);
   (p_event_set)(AW_EVENT_CHAT,
                 Java_vwadm_awa_base_AWChatSensor_chat_handler);
   if (s == JNI_OK) RELEASELOCK(env);
}
```

- Refer to JNI documentation for details of `JNICALL`, `JNIEnv` and so on.
- This function does a lookup on the Java class `vwadm.awa.base.AWChatSensor` to find method `void handleChat(int session, int chattype)`. Function `Java vwadm awa base AWChatSensor nativeConnectToWorld` stores a pointer to this as variable `MID AWChatSensor handleChat`. It then registers the call- back function `Java vwadm awa base AWChatSensor chat handler` to be used by the AW SDK whenever new chat data arrives. This callback uses the vari- able `MID AWChatSensor handleChat` to call the Java method `handleChat`.
- AWAgent is multi=threaded and so any non-reentrant access to variables must be protected. This is done above by the following C macros:

```
#define OBTAINLOCK(env) (*env)->MonitorEnter(env,lockObject)
#define RELEASELOCK(env) (*env)->MonitorExit(env,lockObject)
```

. Any dynamically allocated memory must be released because C has no garbage collection. For global variables, this will require that a native finalize function be called.

4. Compile to a native dynamic link library (DLL). For AWAgent this is done with the lcc compiler (available from http://www.cs.virginia.edu/~lcc-win32/).

5. Extend the `Connection` or `AWConnection` class (see the next section) to load the DLL at start-up time:

```
1  synchronized void loadLibraries()
2  {
3     if (!loaded)
4     {
5        try
6        {
7           System.loadLibrary("awa");
8           loaded = true;
9        }
10       catch (Exception ex)
11       {
12          ex.printStackTrace();
13       }
14    }
15 }
```

References

Austin, J. L. 1962. *How to do things with words*, London: Oxford University Press.

Cohen, P. R. & Levesque, H. J. 1995. Communicative actions for artificial agents, *in* V. Lesser (ed.), *Proceedings First International Conference on Multi-Agent Systems*, Cambridge MA, London: MIT Press, pp. 65-72.

Cohen, P. R. & Perrault, C. R. 1979. Elements of a plan based theory of speech acts, *Cognitive Science* **3**(3): 177-212.

FIPA 2002. Agent communication language specifications. Version H. http://www.fipa.org/specs/fipa00029/SC00029H.html

Liang, S. 1999. *The Java Native Interface: Programmer's Guide and Specification*, Addison-Wesley.

Searle, J. R. 1969. *Speech Acts*, Cambridge: Cambridge University Press.

Appendix E. Survey Questionnaire

Jisc bos

VWADM: An Architecture-Inspired Method for Designing VW Applications

p. 1

Please complete sections 1 and 2 by rating the statements that are associated with each one of them.

Add item

1 Based on your review of the features and capabilities of the VWADM, rate the following statements about its fitness for creating place models of VW applications during the design process.

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
The VWADM is made up of components that may be considered independent (or nearly independent) from each other, but work together during the design process	▭	▭	▭	▭	▭
The VWADM is AT LEAST ONE of the following: (1) adaptable, (2) extensible or (3) customisable	▭	▭	▭	▭	▭
The VWADM is AT LEAST ONE of the following: (1) open to inspection (2) open to modification or (3) open to reuse	▭	▭	▭	▭	▭
The VWADM originates from (or is inspired by) a field or environment in which design (i.e., software or non-software design) is routinely practiced	▭	▭	▭	▭	▭
The VWADM originates from (or is inspired by) a field that is different from software engineering	▭	▭	▭	▭	▭
The VWADM is interesting	▭	▭	▭	▭	▭
AT LEAST ONE of the following can be said about the VWADM: (1) it is compact, (2) it is simple, (2) its use is transparent, (3) its behaviour is transparent or (5) it has clarity of representation	▭	▭	▭	▭	▭

Add item

Add item

2 Based on your review of the features and capabilities of the VWADM, rate the following statements about its utility for creating place models of VW applications during the design process (the statements assume that developers are involved in the design process).

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
Using the VWADM would enable developers to create place models of these applications more quickly during design	▭	▭	▭	▭	▭
Using the VWADM would improve the performance of developers in creating place models of these applications during design	▭	▭	▭	▭	▭
Using the VWADM would increase the productivity of developers when creating place models of these applications during design	▭	▭	▭	▭	▭
Using the VWADM would enhance the effectiveness of developers in creating place models of these applications during design	▭	▭	▭	▭	▭
Using the VWADM would make it easier for developers to create place models of these applications during design	▭	▭	▭	▭	▭
Developers would find the VWADM useful for creating place models of VW applications during design	▭	▭	▭	▭	▭

Add item

Add item

p. 2

Please provide a bit of information about yourself and your design and development experience with regards to VWs (or VW-related platforms such as video games) by answering the questions in sections 3 and 4.

Add item

3 How long have you been involved in the design and development of content in VWs or VW-related platforms (e.g., video games)?

Less than 1 year

1 year or more, but less than 5 years

5 years or more, but less than 10 years

Show all (5)

Add item

Add item

4 Which of the following VW platforms do you have experience using for creating content?

Active Worlds

Second Life

Open Simulator

Show all (7)

Add item

Add item

p.3

Add item

All done. Thank you!

Add item

Appendix F. Evaluation of the VWADM (Survey Data)

Partipant ID	Decomposability	Malleability	Openness	Embedment in a System	Novelty
1	Strongly Agree	Strongly Agree	Agree	Strongly Agree	Strongly Agree
2	Agree	Agree	Agree	Neither Agree nor Disagree	Agree
3	Agree	Agree	Agree	Agree	Agree
4	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree
5	Agree	Agree	Agree	Disagree	Neither Agree nor Disagree
6	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree
7	Agree	Agree	Agree	Agree	Agree
8	Agree	Agree	Agree	Agree	Agree
9	Agree	Agree	Agree	Agree	Agree
10	Strongly Agree	Agree	Agree	Strongly Agree	Disagree
11	Agree	Agree	Agree	Agree	Agree
12	Strongly Agree	Agree	Strongly Agree	Agree	Agree
13	Agree	Strongly Agree	Strongly Agree	Agree	Agree

Interestingness	Elegance	Usefulness (create models more quickly)	Usefulness (improve developer performance)
Agree	Agree	Agree	Agree
Agree	Agree	Neither Agree nor Disagree	Neither Agree nor Disagree
Neither Agree nor Disagree	Agree	Neither Agree nor Disagree	Neither Agree nor Disagree
Strongly Agree	Strongly Agree	Strongly Agree	Neither Agree nor Disagree
Agree	Agree	Neither Agree nor Disagree	Neither Agree nor Disagree
Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree
Agree	Agree	Agree	Disagree
Agree	Agree	Neither Agree nor Disagree	Disagree
Strongly Agree	Agree	Neither Agree nor Disagree	Neither Agree nor Disagree
Agree	Disagree	Neither Agree nor Disagree	Disagree
Agree	Agree	Agree	Agree
Strongly Agree	Agree	Strongly Agree	Strongly Agree
Strongly Agree	Agree	Strongly Agree	Neither Agree nor Disagree

Usefulness (increase developer productivity)	Usefulness (enhance developer effectiveness)	Usefulness (easier to create models)
Neither Agree nor Disagree	Agree	Strongly Agree
Neither Agree nor Disagree	Agree	Agree
Neither Agree nor Disagree	Agree	Strongly Agree
Strongly Agree	Agree	Strongly Agree
Neither Agree nor Disagree	Agree	Agree
Strongly Agree	Agree	Strongly Agree
Disagree	Neither Agree nor Disagree	Neither Agree nor Disagree
Disagree	Neither Agree nor Disagree	Neither Agree nor Disagree
Neither Agree nor Disagree	Neither Agree nor Disagree	Neither Agree nor Disagree
Strongly Disagree	Disagree	Neither Agree nor Disagree
Agree	Agree	Agree
Strongly Agree	Strongly Agree	Strongly Agree
Neither Agree nor Disagree	Agree	Strongly Agree

Usefulness (useful to developers for modelling)	Experience (years)	Active Worlds	Second Life	Open Simulator	High Fidelity
Agree	15 years or more	Selected	Selected	Selected	Not selected
Agree	10 years or more, but less than 15 years	Not selected	Selected	Selected	Selected
Agree	10 years or more, but less than 15 years	Selected	Selected	Selected	Not selected
Strongly Agree	10 years or more, but less than 15 years	Selected	Selected	Selected	Selected
Neither Agree nor Disagree	5 years or more, but less than 10 years	Not selected	Selected	Not selected	Not selected
Strongly Agree	15 years or more	Selected	Selected	Selected	Selected
Agree	5 years or more, but less than 10 years	Not selected	Selected	Not selected	Not selected
Neither Agree nor Disagree	5 years or more, but less than 10 years	Not selected	Selected	Not selected	Not selected
Neither Agree nor Disagree	5 years or more, but less than 10 years	Not selected	Selected	Not selected	Not selected
Neither Agree nor Disagree	5 years or more, but less than 10 years	Not selected	Selected	Not selected	Not selected
Agree	15 years or more	Selected	Selected	Selected	Selected
Strongly Agree	10 years or more, but less than 15 years	Selected	Selected	Selected	Selected
Strongly Agree	10 years or more, but less than 15 years	Not selected	Selected	Not selected	Not selected

Meshmoon	Sansar	Other VW Platform(s)
Not selected	Selected	Not selected
Not selected	Selected	Not selected
Not selected	Selected	Not selected
Selected	Selected	Selected
Not selected	Selected	Selected
Selected	Selected	Selected
Not selected	Selected	Selected
Not selected	Selected	Selected
Not selected	Selected	Not selected
Not selected	Selected	Not selected
Selected	Selected	Selected
Selected	Selected	Selected
Not selected	Selected	Not selected

Appendix G. Participant Information Sheet

[Redacted from this version]

Appendix H. Participant Consent Form

[Redacted from this version]

Appendix I. Summary of the Features and Capabilities of the VWADM

A Summary of the Features and Capabilities of the VWADM

The VWADM is an architecture-inspired method that has been developed for designing VW applications[1]. It combines a generative design grammar (GDG) framework and a generative design agent (GDA) model to form the template of a mechanism that developers can use to create place models of VW applications during the design process. The GDG framework is used to develop GDGs for capturing spatial data that pertain to VW applications, transforming and storing such data and subsequently using it to generate conceptual models of these applications. In complement, the GDA model is used to develop GDAs for automatically generating physical models of VW applications, whose specifications are derived from the GDGs.

1. The GDG Framework

The GDG framework provides guidelines and strategies for developing GDGs. It allows us to specify the general structure of a GDG and its basic components, which are design rules. Figure 1 is an illustration of the GDG framework.

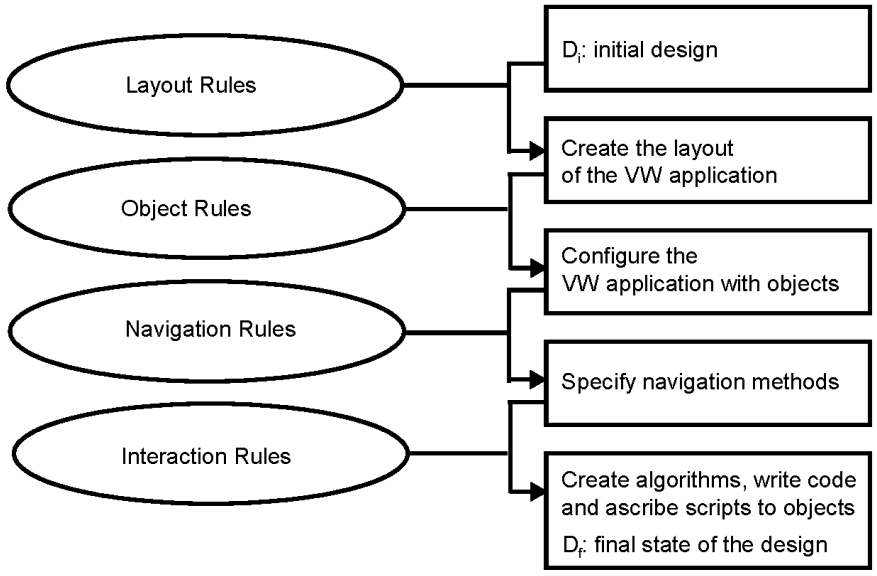


Figure 1. The GDG Framework

By using the GDG framework, GDGs can be developed for creating place-based models of VW applications that are intended to provide support for various in-world activities, and therefore reflect many different architectural styles.

A GDG is a set of rules that describe an architectural style. A GDG G is comprised of design rules R, an initial design D_i, and a final state of the design D_f such that:

$$G = \{R, D_i, D_f\}.$$

[1] A VW application is a place in a VW (e.g., office, library or the area of a map in a video game) where users perform their in-world activities.

The basic components of a GDG are its design rules R. The general structure of a GDG for creating placed-based conceptual models of VW applications consists of four sets of design rules, which are (1) layout rules R_a, (2) object rules R_b, (3) navigation rules R_c, and (4) interaction rules R_d such that:

$$R = \{R_a, R_b, R_c, R_d\}.$$

The four sets of design rules correspond to the four phases for place-based modelling of VW applications, which are as follows:

Layout design: creating the layout of the VW application, where each area has a purpose that accommodates a certain set of intended activities.

Object design: configuring the VW application with objects that provide visual boundaries of the application, as well as visual cues for supporting the intended activities.

Navigation design: specifying navigation methods that use wayfinding aids such as hyperlinks and teleportation devices for assisting the movements of users (using avatars) between different areas of the VW application.

Interaction design: designing algorithms, writing code, and ascribing scripts to objects, to enable users to be able to interact with the VW application.

A GDG's design rule specifies that when the left-hand side object (LHO) is recognised in a VW and the state label (sL) matches, the right-hand side object (RHO) will replace the LHO such that:

$$LHO + sL \rightarrow RHO.$$

State labels direct and ensure that the model of a VW application satisfies its design requirements.

In the following sections, we will use the example of a school application to further explain how a GDG's design rules can be used for creating place-based models of VW applications. For the purpose of creating the layout of the school application, the symbol  will be used to represent the main building (i.e., the initial design) and the symbol  will represent a set of stairs that lead into the main building. The symbol  will represent a classroom of which there are two that are located to the back of the main building and the symbol + will represent the registration mark.

1.1. Layout Rules

The first set of rules to be applied using a GDG are layout rules, which create the layout of the VW application according to the type of activity it is intended to support.

Figure 2 shows that after applying the GDG's layout rules, the RHO (i.e., the layout of the main building with a set of stairs attached to its front) will replace the LHO (i.e., the layout of the main building).

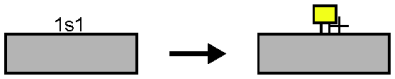


Figure 2. Layout rule for adding a set of stairs to the school's main building

The state label sL = 1 shows that we are working in the first phase of the GDG's design rules and sL= s1 is the design context, which indicates that the layout of a set of stairs, which lead into the main building, should be generated based on a certain specification s1.

Applying the layout rule for adding a set of stairs to the school's main building requires the following conditions to be met:

- The layout of the school's main building  is recognised in the VW
- The design context (s1) matches some current design requirements or needs that are supposed to be used by designers (i.e., manually) or computational agents (i.e., automatically) to model the school application

Figure 3 shows that after applying the GDG's layout rules, the RHO (i.e., the layout of the main building with a set of stairs attached to its front and a classroom added to the back of it) will replace the LHO (i.e., the layout of the main building with a set of stairs attached to it).

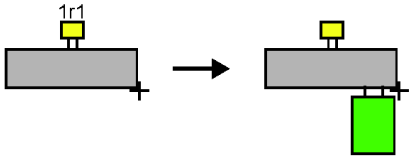


Figure 3. Layout rule for adding a classroom to the back of the main building

The state label sL = 1 shows that we are working in the first phase of the GDG's design rules and sL= r1 is the design context, which indicates that the layout of a room (i.e., a room whose purpose is to support learning activities) should be generated based on a certain specification r1.

Applying the layout rule for adding a classroom to the back of the main building requires the following conditions to be met:

- A layout that represents the school's main building registered with a set of stairs is recognised in the VW
- The design context (r1) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school application

1.2. Object Rules

The second set of rules to be applied using a GDG are object rules, which are used to configure the VW application with various objects that form visual boundaries as well as visual cues of the application for supporting the intended learning activities.

Figure 4 shows that after applying the GDG's object rules for one of the classrooms of the school application, a visual boundary of it is generated.

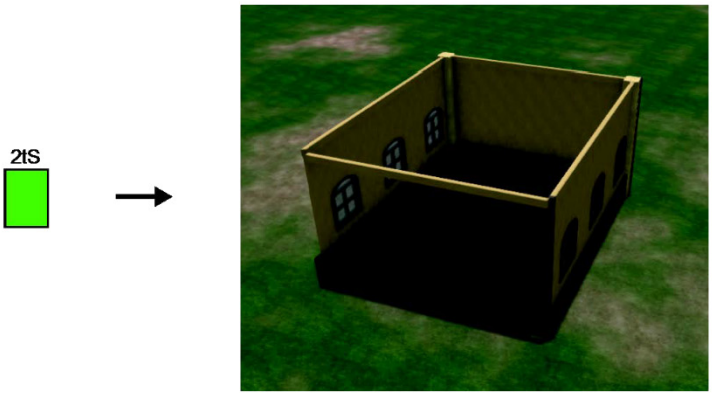


Figure 4. Object rule for configuring the visual boundary of a classroom

The state label sL = 2 shows that we are working in the second phase of the GDG's design rules and sL = tS is the design context, which indicates that a certain texture scheme should be applied to the walls of the classroom.

Applying the object rule for configuring the visual boundary of the classroom requires the following conditions to be met:

- The layout that was generated for the school application contains a main building with a set of stairs that is registered with a classroom to the back of it
- The design context (tS) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school application

Figure 5 shows that after applying the GDG's object rules to one of the classrooms of the school application, it is also configured with visual cues (i.e., a blackboard, book case, books, chairs and desks) for supporting the intended learning activities.

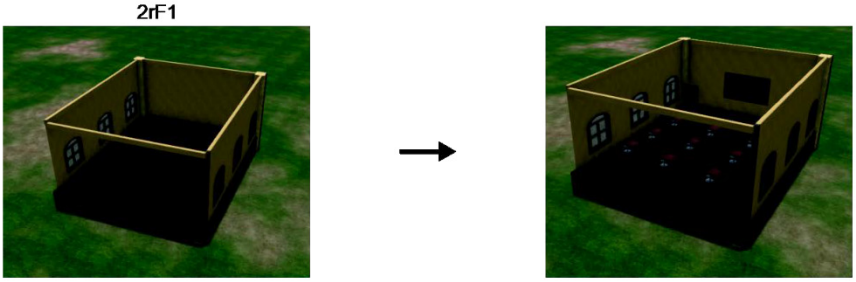


Figure 5. Object rule for configuring a classroom with visual cues

The state label sL = 2 shows that we are working in the second phase of the GDG's design rules and sL = rF1 is the design context, which indicates that a set of furniture should be generated in the classroom.

Applying the object rule for configuring the classroom with visual cues that support learning requires the following conditions to be met:

- A visual boundary of the classroom has been generated
- The design context (rF1) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school application

1.3. Navigation Rules

The third set of rules to be applied using a GDG are navigation rules, which are used to specify the VW application's navigation methods using wayfinding aids such as hyperlinks and teleportation devices.

Figure 6 shows that after applying the GDG's rule, the RHO (i.e., the model of the school with teleportation enabled between its two classrooms) will replace the LHO (i.e., the model of the school without teleportation enabled between its two classrooms).

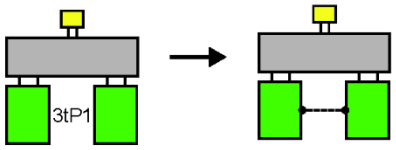


Figure 6. Navigation rule that specifies a teleport navigation method

The state label sL = 3 shows that we are working in the third phase of the GDG's design rules and sL = tP1 is the design context, which indicates that teleportation devices should be generated in order to enable movement from one classroom to the other.

Applying the navigation rule that specifies the school application's navigation method requires the following conditions to be met:

- Visual boundaries of the two classrooms have been generated
- The two classrooms have been configured with all necessary visual cues
- The design context (tP1) matches some current design requirements or needs that are supposed to be used by designers or computational agents to model the school

Figure 7 shows the effect of applying the navigation rule, which generates a teleportation device in this example.

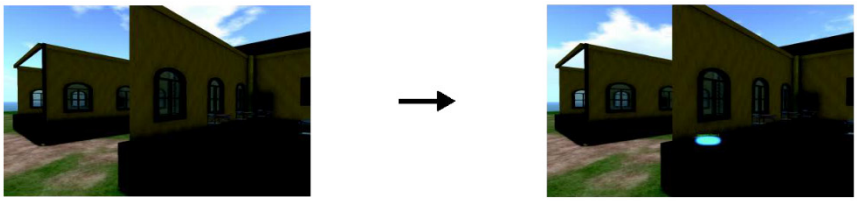


Figure 7. Example of the navigation rule that specifies a teleport navigation method

Both of the images above show the visual boundaries of the two classrooms of the school application. The image on the left is of the two classrooms prior to applying the GDG's navigation rules. The image on the right shows the two classrooms after applying the GDG's navigation rules, which has generated teleportation devices. By ascribing the appropriate scripted behaviour to the teleportation devices, users of the school application will be able to teleport between the two classrooms by clicking on them in alternation.

1.4. Interaction Rules

The fourth and final set of rules to be applied using a GDG are interaction rules, which are used for designing algorithms, writing code and ascribing scripts to objects to enable users to be able to interact with the VW application. As mentioned earlier, interaction rules are non-spatial and non-visual. Therefore, they are expressed using IF... THEN... statements. The following is an example of the IF... THEN... statements for the teleportation device in the school application:

sL = 4
IF: Visual boundaries of the two classrooms have been generated
AND
The two classrooms have been configured with all necessary visual cues
THEN: render teleport device
AND
Ascribe appropriate behaviour to teleport device according to design requirements

Figure 8 shows a physical model of a VW application, which was constructed based on the conceptual model of the school application that was generated by applying the GDG's four sets of design rules.

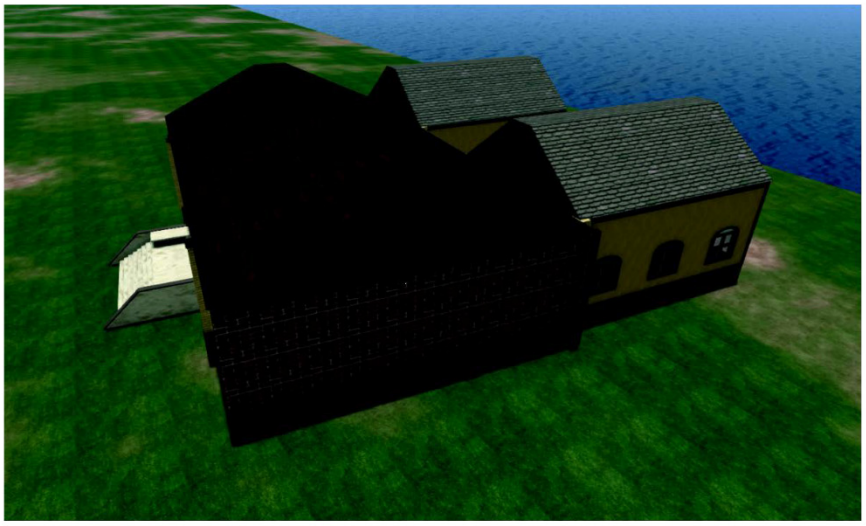


Figure 8. A VW application for supporting learning activities

The GDG framework provides guidelines and strategies for developing GDGs. It allows us to specify the general structure of a GDG and its basic components, which are the design rules. By using the GDG framework, GDGs can be developed for creating place-based models of VW applications with varying degrees of complexity, reflecting different architectural styles. For example, GDGs can be developed for creating models of VW applications such as a building with many differently shaped rooms that span multiple storeys. These models can then be reviewed by stakeholders. In order to create physical models of VW applications, we can introduce the use of GDAs into the design process. A possible workflow would then be to firstly develop a set of GDGs that can be used to generate a description of some VW application, which is also its design specification. Next, a GDA would read the design specification and use some form of reasoning to generate a physical model of the VW application.

2. The GDA Model

The GDA model is used to develop GDAs, which are computational agents that operate in a VW and have the capability to reason about the environment in which they exist. A GDA is comprised of a reasoning mechanism that uses sensors and effectors as an interface between the GDA itself and the VW, as well as for generating physical models of VW applications based on the conceptual models produced by GDGs (i.e., the design specifications). Figure 9 shows the GDA model.

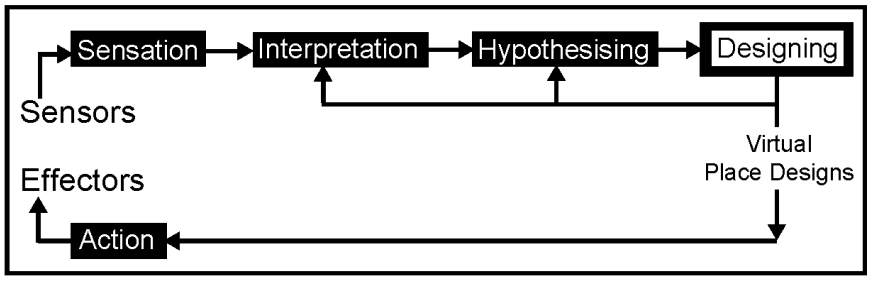


Figure 9. The GDA model

Each GDA wraps around a GDG and senses and effects the properties of the models of VW applications. In addition, a GDA is capable of reasoning about the condition of its environment and act based on condition-action rules. The GDA's reasoning mechanism has five computational processes, which are as follows:

Sensation - using sensors to retrieve raw data from the VW to prepare for the process of interpretation.

Interpretation - interpreting the current design needs and the current state of the VW.

Hypothesising - setting up design goals that aim to eliminate mismatches between the current design needs in the VW and the current state of the VW.

Designing - reading the output of the GDG (i.e., the design specification) and using its description to provide the design of the VW application in order to satisfy current design goals.

Action - planning actions for implementing the design specification in the VW, as well as activating the planned actions in the VW.

A GDA's five computational processes form a recursive loop, in which new creations and changes in the VW trigger GDAs to start a new cycle of reasoning and designing. This way, objects in the VW can be modelled and generated dynamically as needed.

2.1. Sensation

During the sensation process (see Figure 10), the GDA retrieves raw data from the external world to prepare for the interpretation process, in which sense data is transformed in order to construct the GDA's interpreted world W_{int} such that:

$$W_{ext} \rightarrow W_{int}$$

$$W_{ext} = A_{ext} \cup E_{ext} \cup wA_{ext} \cup O_{ext}$$

$$W_{int} = A_{int} \cup E_{int} \cup wA_{int} \cup O_{int}$$

In addition, the GDA interprets the current design needs N in the VW and the current state sT of the VW such that:

$$N = \tau(W_{int})$$

$sT \in W_{int}$

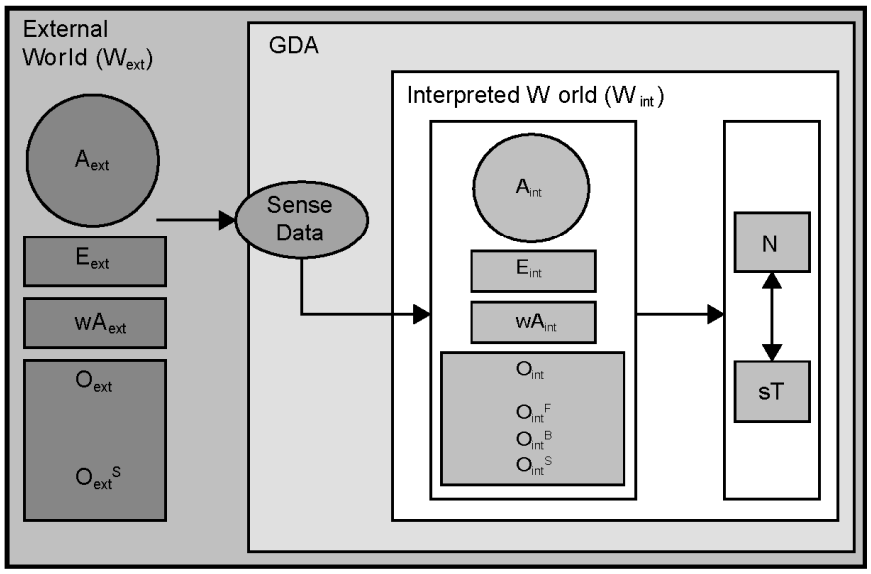


Figure 10. The GDA's sensation process

2.2. Interpretation

During the interpretation process (see Figure 11), the interpreted world W_{int} is constructed based on the external world W_{ext} such that:

$$W_{ext} \rightarrow W_{int}$$

$$W_{ext} = A_{ext} \cup E_{ext} \cup wA_{ext} \cup O_{ext}$$

$$W_{int} = A_{int} \cup E_{int} \cup wA_{int} \cup O_{int}$$

The GDA's interpretation process has three sub-processes. In the first sub-process, the sense data from the external world W_{ext} is transformed into information that can be understood by the GDA such that:

$$A_{int} = \tau(A_{ext})$$

$$E_{int} = \tau(E_{ext})$$

$$wA_{int} = \tau(wA_{ext})$$

$$O_{int\_i}^{S} = \tau(O_{ext\_i}^{S})$$

In the second sub-process, for any object in the VW, the interpreted functions $O_{int\_i}^{F}$ and interpreted behaviours $O_{int\_i}^{B}$ are derived from the interpreted structures $O_{int\_i}^{S}$ such that:

$$O_{int\_i}^{B} = \tau(O_{int\_i}^{S})$$

$$O_{int\_i}^{F} = \tau(O_{int\_i}^{B})$$

In the third sub-process, the current design needs N in the VW and the current state sT of the VW are interpreted based on the information gained from the first two processes such that:

$$N = \tau(W_{int})$$

$$sT \in W_{int}$$

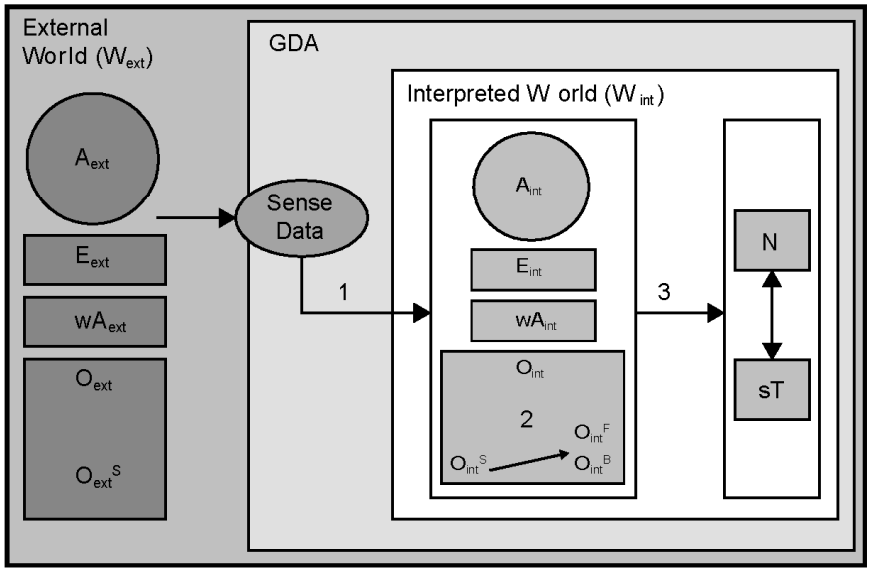


Figure 11. The GDA's interpretation process

2.3. Hypothesising

During the hypothesising process (see Figure 12), the GDA sets up design goals in the expected world W_{exp} in order to eliminate mismatches between the current design needs N in the VW and the current state sT of the VW such that:

$$W_{int} \rightarrow W_{exp}$$

$$W_{exp} = A_{exp} \cup E_{exp} \cup O_{exp}$$

It is optional for an object in the GDA's expected world to have a counterpart in the interpreted world, since objects in VWs can be generated based either on existing structures or by new creations.

The GDA is capable of hypothesising three different types of goals. The first type are goals that are related to designing objects in VWs, which are represented by expected functions O_{exp}^{F} and expected behaviours O_{exp}^{B} such that:

$$O_{exp}^{F} = \{O_{exp\_1}^{F}, O_{exp\_2}^{F}, \ldots, O_{exp\_n}^{F}\}$$

$$O_{exp}^{B} = \{O_{exp\_1}^{B}, O_{exp\_2}^{B}, \ldots, O_{exp\_n}^{B}\}$$

for any expected function $O_{exp\_i}^{F}$ and expected behaviour $O_{exp\_i}^{B}$ such that:

$$O_{exp\_i}^{F} \in \tau(N, sT)$$

$$O_{exp\_i}^{B} = \tau(O_{exp\_i}^{F})$$

The second type of goals that the GDA is capable of hypothesising are those that are used for initiating changes of properties of avatars in the VW. The third type of goals that the GDG is capable of hypothesising are those that are used for initiating events in the VW such that:

$$A_{exp} \in \tau A(N, sT)$$

$$E_{exp} \in \tau E(N, sT)$$

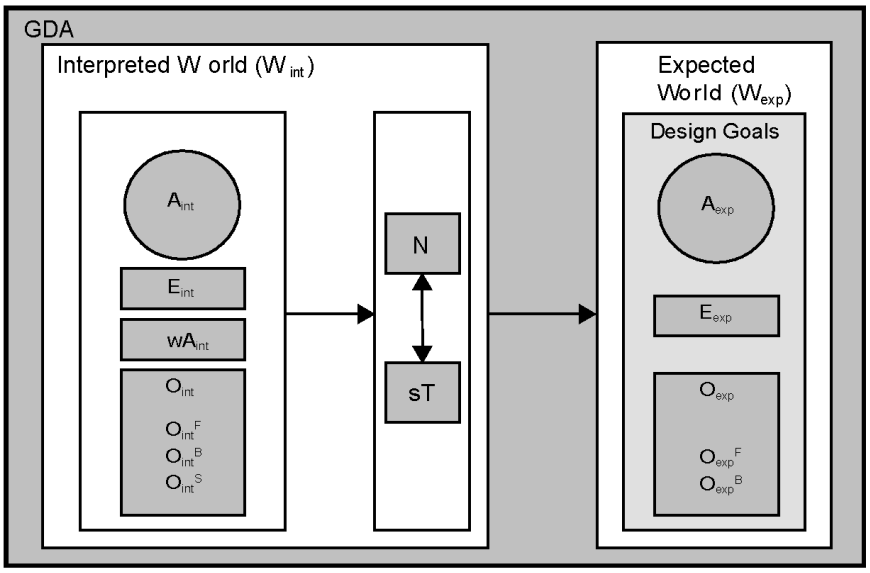


Figure 12. The GDA's hypothesising process

2.4. Designing

During the designing process, the GDA generates an object in order to satisfy a set of requirements or design goals. The object is represented by O_{exp}^{S} such that:

$$O_{exp}^{S} = \tau(O_{exp}^{F}, O_{exp}^{B})$$

The GDA's generative capability enables objects to be generated dynamically as needed. The GDA's design component is supported by the application of a GDG in order to generate the objects in the VW.

2.5. Action

During the action process (see Figure 13), the GDA carries out action planning and action activation. In action planning, the GDA plans actions for generating the model of an object O_{exp}^{S} based on the GDG's specification and for realising other initiated changes A_{exp} and E_{exp} in the VW. In action activation, the planned actions are activated in the VW by the GDA through the use of its effectors.

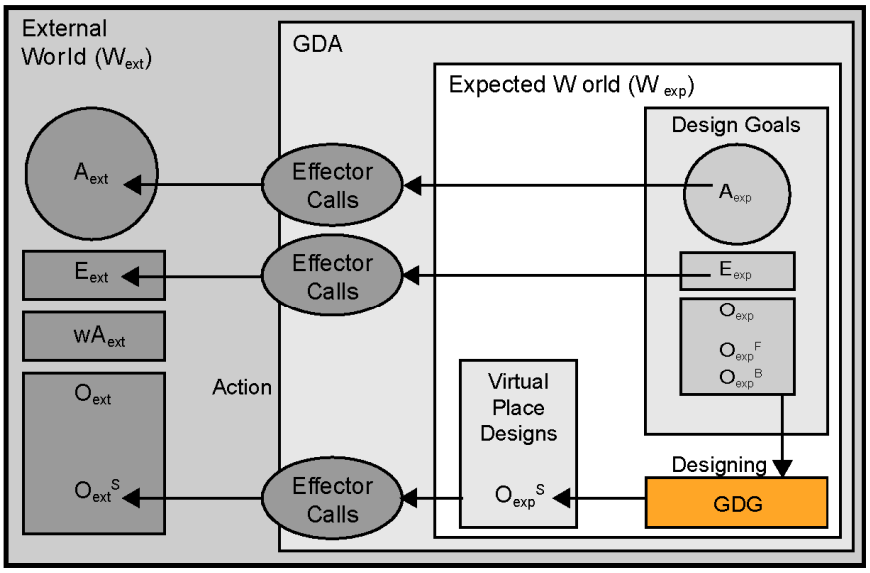


Figure 13. The GDA's design and action processes

The GDA model is a computational agent model that operates in a VW and has computational processes for reasoning, which can be used to automatically generate place-based models of VW applications. GDAs are wrappers for GDGs and use sensors and effectors to dynamically generate objects in VWs. These objects may be a set of assets for modelling VW applications or they may be the VW application itself (including the objects that comprise it). A GDA is capable of reasoning about the condition of its environment and act based on condition-action rules, which it applies recursively until it meets its design goals.

3. Proof of Concept

The AW agent package is an implementation of the VWADM's reasoning mechanism for designing VW applications. It uses Jess and Java sensors and effectors for coding agent reasoning as rules. In addition, it uses an SDK called the AW SDK, which agents rely on for communicating with Active Worlds servers.

3.1. Jess

Jess is a small, light and fast rule engine and scripting environment for developing Java software that has the capacity to reason using knowledge supplied in the form of declarative rules. It is generally used for the automation of expert systems as well as to develop intelligent agent systems.

The Jess rule engine employs the Rete algorithm for rapid processing of rules and it can be used as a means to efficiently create Java applets and servlets and Enterprise JavaBeans.

Jess' scripting environment allows the use of either the Jess language, which is a highly specialised form of Lisp, or the Java programming language for coding rules. Jess enables the integration of both the Jess language and Java in such a way that Java functions can be accessed via the Jess language and vice versa.

Jess is written in Java and requires a Java Virtual Machine (JVM) to function. As such, both the Jess language and Java are native programming languages for it.

3.2. Java

Java is a general-purpose computer programming language that is concurrent, class-based and object-oriented. It is used to develop portable software that can run securely on any platform that supports it without the need for recompilation.

As a result of its portability and security features, Java is well-suited for developing computer programs that operate in networked environments. Since the agents that were implemented in this research communicate over a network with an Active Worlds server, Java provides the perfect companion for Jess, whose state could be packaged up at an origin, sent across a wire and reconstituted at the destination.

3.3. AW SDK

The AW SDK is a set of software development tools that provide an API for developing client applications, which can communicate with Active Worlds servers and function within the VWs generated by these servers. The core component of the AW SDK is the aw.dll file, which is a Microsoft Windows dynamic-link library (DLL) file that implements the Active Worlds client-server protocol.

In order to develop an application using the AW SDK, a programmer simply writes a C program which includes the header file aw.h and links to the import library aw.lib. The compiled executable can be run on any computer from anywhere as long as that computer has a network connection to the Active Worlds servers and aw.dll is available on it.

The AW SDK can be used to develop applications such as an automated program that explores and creates objects in VWs generated by Active Worlds servers. It can also be used to develop bots or non-playable characters (NPCs), which are typically avatars that are controlled by computer programs rather than humans. Administrators may also use the AW SDK to develop utilities for managing their VWs in Active Worlds.

3.4. Overall Design of the AW Agent Package

The AW agent package was designed to provide a flexible framework in which agents are the basis for all of the elements of a VW in Active Worlds. By using intelligent agent models, it becomes possible to dynamically generate such elements. The AW agent package is based on a set of abstract classes that form the generic architecture for constructing a multi-agent system (MAS) of agents. A MAS is an aggregation of agents that share a common connection with a VW. Figure 14 shows an entity relationship diagram (ERD) that depicts the concept of a MAS and agents.

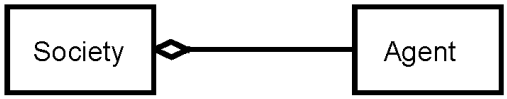


Figure 14. MAS and agents

3.5. Main Components of the AW Agent Package

Agents usually share some ontological connection such as a room agent plus a set of wall agents that collectively comprise a virtual conference room. In complement, the MAS manages computational resources such as the connection to the VW, on behalf of the agents. Each entity that is capable of reflexive, reactive or reflective behaviours is modelled as an agent. Such an agent inherits these three generic behaviours (including an optional 3D representation) and can dynamically change the 3D representation and generate non-visual behaviours. The procedure to assert an object into a VW in Active Worlds is to copy an existing object, move it to a desired location and edit a dialog box to specify its properties. The procedure to create an agent in the VW is to configure it as a set of sensors and a set of rules. An agent can be configured using an XML file that is loaded through the use of a validating parser, rather than being statically linked at compile time. This flexible method enables the reconfiguration of agents running in the VW without having to recompile and restart the server. Figure 15 shows the components that comprise an agent (or a ReteAgent).

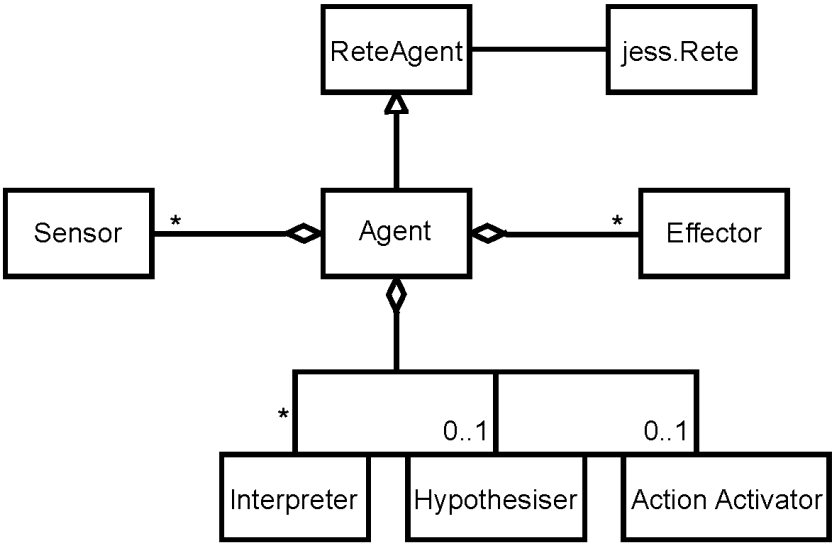


Figure 15. Components of an agent/ReteAgent

3.6. Architecture of the AW Agent Package

The VWADM's AW agent package implements sensors as java objects that sense an agent's environment. Effectors act on the environment. Both sensors and effectors use a Java Native Interface (JNI) to the AW SDK. An effector is a function call from the right-hand side of a rule. ReteAgent is an implementation of an

agent that uses Jess for everything except sensors and effectors. Figure 16 shows the architecture of ReteAgent.

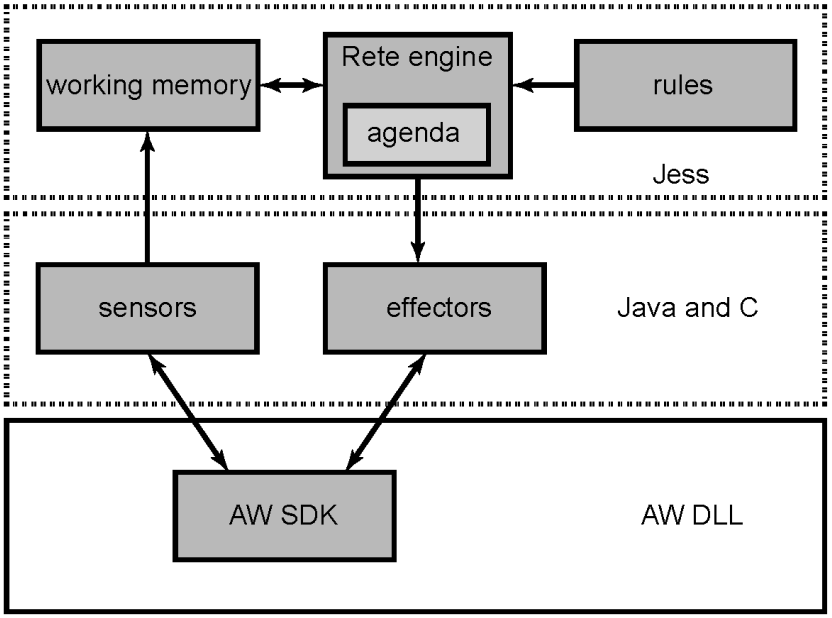


Figure 16. Architecture of an agent/ReteAgent

3.7. Basic Functionality of the AW Agent Package

A MAS can contain one or more agents that are instances of ReteAgent. The creator of a ReteAgent configures the agent by specifying a set of sensors, whose data is stored in Jess' working memory. Sensors also record messages from other agents that is stored in Jess' working memory. Each time new sense data is received, a new fact in the form of a Java bean is asserted into Jess' working memory. Figure 17 shows an example of how a MAS communicates with the Active Worlds server.

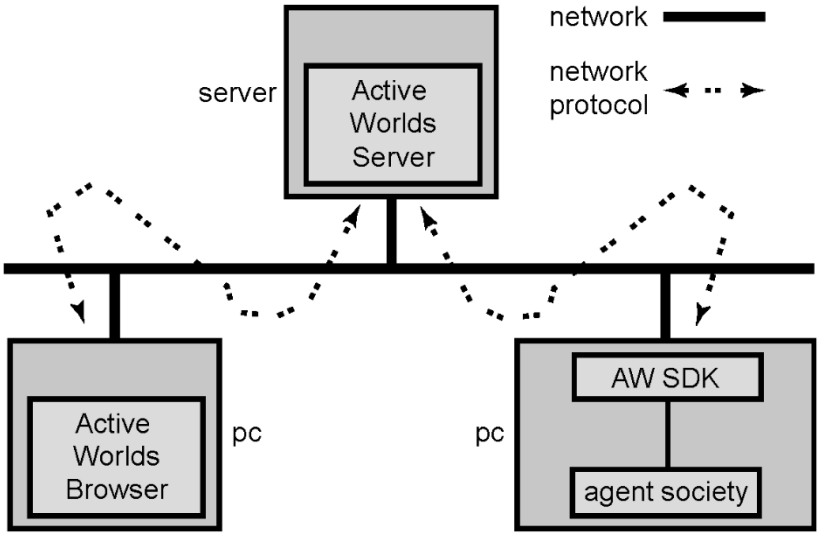


Figure 17. MAS communication

3.8. Guides for Implementing and Extending the Functionality of Agents using the AW Agent Package

The AW agent package can be used to implement agents using Java sensors and effectors and the Jess language for coding their reasoning as rules in Jess. A basic user guide is available, which discusses the implementation of the sensors and effectors of such agents. The sensors and effectors are class-based. Therefore, developers can write their own sensors and effectors by extending the classes in the vwadm.awa.base Java package. An advanced user guide that discusses how to do so is also available. The advanced user guide provides additional instructions on how to extend the functionality of agents to enable them to communicate with each other as well as for them to be able to dynamically create other agents. Such functionality is important to a MAS because it enables agents to enlist the support of other agents to achieve goals, commit to the execution of actions and report on progress.

This page is intentionally left blank.